Питер Абель

АССЕМБЛЕР И ПРОГРАММИРОВАНИЕ ДЛЯ IBM PC

Технологический институт Британская Колумбия

Peter Abel

IBM PC ASSEMBLER LANGUAGE AND PROGRAMMING

British Columbia Institute of Technology

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ ПЕРЕВОДЧИКА

Книга представляет собой учебник по программированию на языке Ассемблера для персональных компьютерах, совместимых с IBM PC, адресованный прежде всего начинающим. Обилие примеров и исходных текстов программ представляет несомненное достоинство книги, позволяющее начинать практическое программирование уже с первых страниц книги. Профессиональные программисты смогут найти в книге много полезной информации. Стиль книги очень живой, простой, не требующий никакой специальной или математической подготовки. Единственное, что необходимо для работы над книгой, - это

постоянный доступ к персональному компьютеру.

Переводчик в основном придерживался терминологии книг В.М.Брябрина "Программное обеспечение персональных ЭВМ" (1988), С.Писарева, Б.Шура "Программно-аппаратная организация компьютера IBM PC" (1987), В.Л.Григорьева "Программирование однокристальных микропроцессоров" (1987), а также А.Б.Борковского "Англо-русский словарь по программированию и информатике" (1987). Во многих случаях переводчик придерживался "профессионального диалекта" максимально щадящего технические термины в оригинале. Такой диалект принят во многих коллективах программистов-разработчиков, где чаще всего приходится работать с оригинальной документацией на английском языке, ввиду острейшего дефицита отечественной литературы по данной тематике.

Большинство примеров, приведенных в данной книге, проверены на компьютерах совместимых с IBM PC. При переводе без специальных оговорок исправлены мелкие неточности и опечатки оригинала.

Текст перевода сформирован и отредактирован в интегрированной системе Framework.

Автор перевода благодарен всем, кто оказал помощь при вводе рукописи на машинные носители. Особую признательность автор перевода выражает своей жене.

ПРЕДИСЛОВИЕ

Появление микропроцессоров в 60-х годах связано с разработкой интегральных схем (ИС). Интегральные схемы объединяли в себе различные электронные компоненты в единый элемент на силиконовом "чипе". Разработчики установили этот крошечный чип в устройство, напоминающие сороконожку и включили его в функционирующие системы. В начале 70-х микрокомпьютеры на процессоре Intel 8008 возвестили о первом поколении микропроцессоров.

К 1974 году появилось второе поколение микропроцессоров общего назначения Intel 8080. Данный успех побудил другие фирмы к производству этих или аналогичных процессоров.

В 1978 году фирма Intel выпустила процессор третьего поколения - Intel 8086, который обеспечивал некоторую совместимость с 8080 и являлся значительным продвижением вперед в данной области. Для поддержки более простых устройств и обеспечения совместимости с устройствами ввода/вывода того времени Intel разработал разновидность процессора 8086 - процессор 8088, который в 1981 году был выбран фирмой iBM для ее персональных компьютеров.

Более развитой версией процессора 8088 является процессор 80188, а для процессора 8086 - процессоры 80186, 80286 и 80386, которые обеспечили дополнительные возможности и повысили мощность вычислений. Микропроцессор 80286, установленный в компьютерах IBM AT появился в 1984 году. Все эти процессоры имеют отношение к развитой архитектуре процессоров фирмы Intel и обозначаются как iAPX 86, iAPX 88, iAPX 86, iAPX286 и iAPX386, где APX - Intel Advanced Processor Architecture.

Распространение микрокомпьютеров послужило причиной пересмотра отношения к языку ассемблера по двум основным причинам. Во-первых, программы, написанные на языке ассемблера, требуют значительно меньше памяти и времени выполнения. Во-вторых, знание языка ассемблера и результирующего машинного кода дает понимание архитектуры машины, что вряд ли обеспечивается при работе на языке высокого уровня. Хотя большинство специалистов в области программного обеспечения ведут разработки на языках высокого уровня, таких как Паскаль или C, что проще при написании программ, наиболее мощное и эффективное программное обеспечение полностью или частично написано на языке ассемблера.

Языки высокого уровня были разработаны для того, чтобы избежать

специальной технической особенности конкретных компьютеров. Язык ассемблера, в свою очередь, разработан для конкретной специфики компьютера или точнее для специфики процессора. Следовательно, для того, чтобы написать программу на языке ассемблера для конкретного компьютера, следует знать его архитектуру и данная книга содержит весь необходимый базовый материал. Для работы кроме этого материала и соответствующих знаний необходимы следующее:

- Доступ персональному компьютеру IBM PC или совместимому с ним с оперативной памятью - минимум 64К и одним дисководом. Лучше, но не обязательно, если будет дополнительная память и второй дисковод или винчестер.
- Знакомство с руководством по IBM PC.
- Дискета, содержащая транслятор с языка ассемблера, предпочтительно, но не обязательно, последней версии.
- Копию операционной системы PC-DOS или MS-DOS, лучше последней версии.

Следующее является не обязательным для данной темы:

- Опыт программирования. Хотя эти знания могут помочь быстрее освоить некоторые идеи программирования, они не обязательны.
- Хорошие знания в электронике или схемотехнике. Данная книга дает всю необходимую информацию об архитектуре PC, которая требуется для программирования на языке ассемблера.

Операционные системы

Назначение операционной системы - позволить пользователю управлять работой на компьютере: вызывать для выполнения конкретные программы, обеспечивать средства для сохранения данных (каталог), иметь доступ к информации на диске.

Основной операционной системой для PC и совместимых моделей является MS-DOS фирмы Microsoft, известная как PC-DOS для IBM PC. Особенности некоторых версий: 2.0 обеспечивает поддержку твердого диска (винчестера), 3.0 применяется в компьютерах AT, 4.0 обеспечивает работу в многопользовательском режиме. Рассмотрение профессиональной операционной системы UNIX и ее аналога для PC XENIX выходит за рамки данной книги.

Подход к книге

Данная книга преследует две цели: она является учебником, а так же постоянным справочным пособием для работы. Чтобы наиболее эффективно восполнить затраты на микрокомпьютер и программное обеспечение, необходимо тщательно прорабатывать каждую главу и перечитывать материал, который не сразу ясен. Ключевые моменты находятся в примерах программ, их следует преобразовать в выполнимые модули и выполнить их. Прорабатывайте упражнения, приведенные в конце каждой главы.

Первые восемь глав составляют базовый материал для данной книги и для языка ассемблера. После этих глав можно продолжить с глав 9, 11, 12, 14, 15, 19, 20 или 21. Связанными являются главы с 8 по 10, 12 и 13, с 15 по 18, главы с 22 по 25 содержат справочный материал.

Когда вы завертшите работу с книгой, вы сможете:

- понимать устройство персонального компьютера;
- понимать коды машинного языка и шестнадцатиричный формат;
- понимать назначение отдельных шагов при ассемблировании, компановке и выполнении;
- писать программы на языке ассемблера для управления экраном,

арифметических действий, преобразования ASCII кодов в двоичные форматы, табличного поиска и сортировки, дисковых операций ввода/вывода;

- выполнять трассировку при выполнении программы, как средство отладки;
- писать собственные макрокоманды;
- компановать вместе отдельные программы.

Изучение языка ассемблера и создание работающих программ - это захватывающий процесс. Затраченное время и усилия несомненно будут вознаграждены.

Признательность автора

Автор благодарен за помощь и сотрудничество всем, кто внес предложения и просматривал рукопись.

ГЛАВА 1 Введение в семейство персональных компьютеров IBM PC

Ц е л ь: объяснить особенности технических средств микрокомпьютера и организации программного обеспечения.

ВВЕДЕНИЕ

Написание ассемблерных программ требует знаний организации всей системы компьютера. В основе компьютера лежат понятия бита и байта. Они являются тем средством, благодаря которым в компьютерной памяти представлены данные и команды.

Программа в машинном коде состоит из различных сегментов для определения данных, для машинных команд и для сегмента, названного стеком, для хранения адресов. Для выполнения арифметических действий, пересылки данных и адресации компьютер имеет ряд регистров. Данная глава содержит весь необходимый материал по этим элементам компьютера, так что вы сможете продвинутся к гл.2 к вашей первой программе на машинном языке.

БИТЫ И БАЙТЫ

Для выполнения программ компьютер временно записывает программу и данные в основную память. Это память, которую люди имеют в виду, когда утверждают, что их компьютер имеет, например, 512К памяти. Компьютер имеет также ряд регистров, которые он использует для временных вычислений.

Минимальной единицей информации в компьютере является бит. Бит может быть выключен, так что его значение есть нуль, или включен, тогда его значение равно единице. Единственный бит не может представить много информации в отличие от группы битов.

Группа из девяти битов представляет собой байт; восемь битов которого содержат данные и один бит - контроль на четность. Восемь битов обеспечивают основу для двоичной арифметики и для представления символов, таких как буква A или символ *. Восемь битов дают 256 различных комбинаций включенных и выключенных состояний: от "все выключены" (00000000) до "все включены" (11111111). Например, сочетание включенных и выключенных битов для представления буквы A выглядит как 01000001, а для символа * - 00101010 (это можно не запоминать). Каждый байт в памяти компьютера имеет

уникальный адрес, начиная с нуля.

Требование контроля на честность заключается в том, что количество включенных битов а байте всегда должно быть не четно. Контрольный бит для буквы А будет иметь значение единица, а для символа * - ноль. Когда команда обращается к байту в памяти, компьютер проверяет этот байт. Если число включенных битов является четным, система выдает сообщение об ошибке. Ошибка четности может явится результатом сбоя оборудования или случайным явлением, в любом случае, это бывает крайне редко.

Может появится вопрос, откуда компьютер "знает", что значения бит 01000001 представляют букву А. Когда на клавиатуре нажата клавиша А, система принимает сигнал от этой конкретной клавиши в байт памяти. Этот сигнал устанавливает биты в значения 01000001. Можно переслать этот байт в памяти и, если передать его на экран или принтер, то будет сгенерирована буква А.

По соглашению биты в байте пронумерованы от 0 до 7 справа налево, как это показано для буквы А:

```
Номера бит:      7 6 5 4 3 2 1 0
Значения бит:     0 1 0 0 0 0 0 1
```

Число 2 в десятой степени равно 1024, что составляет один килобайт и обозначается буквой К. Например, компьютер с памятью в 512К содержит 512 x 1024, т.е. 524288 байт.

Процессор в PC и в совместимых моделях использует 16-битовую архитектуру, поэтому он имеет доступ к 16-битовым значениям как в памяти, так и в регистрах. 16-битовое (двухбайтовое) поле называется словом. Биты в слове пронумерованы от 0 до 15 справа налево, как это показано для букв PC:

```
Номера бит:   15 14 13 12 11 10 9 8  |  7 6 5 4 3 2 1 0
Значения бит:  0  1  0  1  0  0 0 0  |  0 1 0 0 0 0 1 1
```

ASCII КОД

Для целей стандартизации в микрокомпьютерах используется американский национальный стандартный код для обмена информацией ASCII (American National Standard Code for Information Interchange). [Читается как "аски" код (прим. переводчика)]. Именно по этой причине комбинация бит 01000001 обозначает букву А. Наличие стандартного кода облегчает обмен данными между различными устройствами компьютера. 8-битовый расширенный ASCII-код, используемый в PC обеспечивает представление 256 символов, включая символы для национальных алфавитов. В прил.1 приведен список символов ASCII кода, а в гл.8 показано как вывести на экран большинство из 256 символов.

ДВОИЧНЫЕ ЧИСЛА

Так как компьютер может различить только нулевое и единичное состояние бита, то он работает системе исчисления с базой 2 или в двоичной системе. Фактически бит унаследовал свое название от английского "BInary digiT" (двоичная цифра).

Сочетанием двоичных цифр (битов) можно представить любое значение. Значение двоичного числа определяется относительной позицией каждого бита и наличием единичных битов. Ниже показано восьмибитовое число содержащее все единичные биты:

```
Позиционные веса:      128 64 32 16 8 4 2 1
```

Включенные биты: 1 1 1 1 1 1 1 1

Самый правый бит имеет весовое значение 1, следующая цифра влево - 2, следующая - 4 и т.д. Общая сумма для восьми единичных битов в данном случае составит 1 + 2 + 4 + ... + 128, или 255 (2 в восьмой степени - 1).

Для двоичного числа 01000001 единичные биты представляют значения 1 и 64, т.е. 65. Но 01000001 представляет также букву A! Действительно, здесь момент, который необходимо четко уяснить. Биты 01000001 могут представлять как число 65, так и букву A:

- если программа определяет элемент данных для арифметических целей, то 01000001 представляет двоичное число эквивалентное десятичному числу 65;
- если программа определяет элемент данных (один или более смежных байт), имея в виду описательный характер, как, например, заголовок, тогда 01000001 представляет собой букву или "строку".

При програмировании это различие становится понятным, так как назначение каждого элемента данных определено.

Двоичное число неограничено только восемью битами. Так как процессор 8088 использует 16-битовую архитектуру, он автоматически оперирует с 16-битовыми числами. 2 в степени 16 минус 1 дает значение 65535, а немного творческого программирования позволит обрабатывать числа до 32 бит (2 в степени 32 минус 1 равно 4294967295) и даже больше.

Двоичная арифметика

Микрокомпьютер выполняет арифметические действия только в двоичном формате. Поэтому программист на языке ассемблера должен быть знаком с двоичным форматом и двоичным сложением:

```
    0 + 0 = 0
    1 + 0 = 1
    1 + 1 = 10
1 + 1 + 1 = 11
```

Обратное внимание на перенос единичного бита в последних двух операциях. Теперь, давайте сложим 01000001 и 00101010. Букву A и символ *? Нет, число 65 и число 42:

```
Двоичные          Десятичные

01000001              65
00101010              42
--------           ---
01101011             107
```

Проверьте, что двоичная сумма 01101011 действительно равна 107. Рассмотрим другой пример:

```
Двоичные          Десятичные

00111100              60
00110101              53
--------           ---
01110001             113
```

Отрицательные числа

Все представленные выше двоичные числа имеют положительные значения, что обозначается нулевым значением самого левого (старшего) разряда. Отрицательные двоичные числа содержат единичный бит в старшем разряде и выражаются двоичным дополнением. Т.е., для представления отрицательного двоичного числа необходимо инвертировать все биты и прибавить 1. Рассмотрим пример:

```
          Число 65:        01000001
          Инверсия:        10111110
          Плюс 1:          10111111  (равно -65)
```

Если прибавить единичные значения к числу 10111111, 65 не получится. Фактически двоичное число считается отрицательным, если его старший бит равен 1. Для определения абсолютного значения отрицательного двоичного числа, необходимо повторить предыдущие операции: инвертировать все биты и прибавить 1:

```
          Двоичное значение:  10111111
          Инверсия:           01000000
          Плюс 1:             01000001  (равно +65)
```

Сумма +65 и -65 должна составить ноль:

```
                         01000001  (+65)
                         10111111  (-65)
                         --------
                     (1) 00000000
```

Все восемь бит имеют нулевое значение. Перенос единичного бита влево потерян. Однако, если был перенос в знаковый разряд и из разрядной сетки, то результат является корректным.

Двоичное вычитание выполняется просто: инвертируется знак вычитаемого и складываются два числа. Вычтем, например, 42 из 65. Двоичное представление для 42 есть 00101010, и его двоичное дополнение: - 11010110:

```
             65        01000001
           +(-42)       11010110
            -----      --------
             23    (1) 00010111
```

Результат 23 является корректным. В рассмотренном примере произошел перенос в знаковый разряд и из разрядной сетки.

Если справедливость двоичного дополнения не сразу понятна, рассмотрим следующие задачи: Какое значение необходимо прибавить к двоичному числу 00000001, чтобы получить число 00000000? В терминах десятичного исчисления ответом будет -1. Для двоичного рассмотрим 11111111:

```
                         00000001
                         11111111
                         --------
          Результат:     (1) 00000000
```

Игнорируя перенос (1), можно видеть, что двоичное число 11111111 эквивалентно десятичному -1 и соответственно:

```
             0         00000000
           -(+1)       -00000001
            ----       --------
            -1         11111111
```

Можно видеть также каким образом двоичными числами представлены

уменьшающиеся числа:

```
+3    00000011
+2    00000010
+1    00000001
 0    00000000
-1    11111111
-2    11111110
-3    11111101
```

Фактически нулевые биты в отрицательном двоичном числе определяют его величину: рассмотрите позиционные значения нулевых битов как если это были единичные биты, сложите эти значения и прибавьте единицу.

Данный материал по двоичной арифметике и отрицательным числам будет особенно полезен при изучении гл.12 и 13.

ШЕСТНАДЦАТИРИЧНОЕ ПРЕДСТАВЛЕНИЕ

Представим, что необходимо просмотреть содержимое некоторых байт в памяти (это встретится в следующей главе). Требуется определить содержимое четырех последовательных байт (двух слов), которые имеют двоичные значения. Так как четыре байта включают в себя 32 бита, то специалисты разработали "стенографический" метод представления двоичных данных. По этому методу каждый байт делится пополам и каждые полбайта выражаются соответствующим значением. Рассмотрим следующие четыре байта:

```
Двоичное:   0101 1001 0011 0101 1011 1001 1100 1110
Десятичное:    5    9    3    5   11    9   12   14
```

Так как здесь для некоторых чисел требуется две цифры, расширим систему счисления так, чтобы 10=A, 11=B, 12=C, 13=D, 14=E, 15=F. таким образом получим более сокращенную форму, которая представляет содержимое вышеуказанных байт:

```
59   35   B9   CE
```

Такая система счисления включает "цифры" от 0 до F, и так как таких цифр 16, она называется шестнадцатиричным представлением. В табл.1.1. приведены двоичные, десятичные и шестнадцатиричные значения чисел от 0 до 15.

Двоич.	Дес.	Шест.	Двоич.	Дес.	Шест.
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

Табл.1.1. Двоичное, десятичное и шестнадцатиричное представления.

Шестнадцатиричный формат нашел большое применение в языке ассемблера.

В листингах ассемблирования программ в шестнадцатеричном формате показаны все адреса, машинные коды команд и содержимое констант. Также для отладки при использовании программы DOS DEBUG адреса и содержимое байтов выдается в шестнадцатиричном формате.

Если немного поработать с шестнадцатиричным форматом, то можно быстро привыкнуть к нему. рассмотрим несколько простых примеров шестнадцатиричной арифметики. Следует помнить, что после шестнадцатиричного числа F следует шестнадцатиричное 10, что равно десятичному числу 16.

```
6   5   F   F   10  FF
4   8   1   F   10   1
-   -  --  --   --  ---
A   D  10  1E   20  100
```

Заметьте также, что шест.20 эквивалентно десятичному 32, шест.100 - десятичному 256 и шест.100 - десятичному 4096.

В данной книге шестнадцатиричные числа записываются, например, как шест.4В, двоичные числа как дв.01001011, и десятичные числа, как 75 (отсутствие какого-либо описания предполагает десятичное число). Исключения возможны, когда база числа очевидна из контекста. Для индикации шест. числа в ассемблерной программе непосредственно после числа ставится символ "Н", например, 25Н (десятичное значение 37). Шест. число всегда начинается с десятичной цифры 0-9, таким образом, В8Н записывается как 0В8Н.

В прил.2 показано как преобразовывать шестнадцатиричные значения в десятичные и обратно. Теперь рассмотрим некоторые характеристики процессора PC, которые необходимо понять для перехода к гл.2.

СЕГМЕНТЫ

Сегментом называется область, которая начинается на границе параграфа, т.е. по любому адресу, который делится на 16 без остатка. Хотя сегмент может располагаться в любом месте памяти и иметь размер до 64 Кбайт, он требует столько памяти, сколько необходимо для выполнения программы. Имеется три главных сегмента:

1. С е г м е н т к о д о в. Сегмент кодов содержит машинные команды, которые будут выполняться. Обычно первая выполняемая команда находится в начале этого сегмента и операционная система передает управление по адресу данного сегмента для выполнения программы. Регистр сегмента кодов (CS) адресует данный сегмент.
2. С е г м е н т д а н н ы х. Сегмент данных содержит определенные данные, константы и рабочие области, необходимые программе. Регистр сегмента данных (DS) адресует данный сегмент.
3. С е г м е н т с т е к а. Стек содержит адреса возврата как для программы для возврата в операционную систему, так и для вызовов подпрограмм для возврата в главную программу. Регистр сегмента стека (SS) адресует данный сегмент.

Еще один сегментный регистр, регистр дополнительного сегмента (ES), предназначен для специального использования. На рис.1.2 графически представлены регистры SS, DS и CS. Последовательность регистров и сегментов на практике может быть иной. Три сегментных регистра содержат начальные адреса соответствующих сегментов и каждый сегмент начинается на границе параграфа.

```
               ╓===============╖
```

```
            ¦    D O S      ¦
    --------¬  ¦- - - - - - - - ¦ ¬
 SS ¦ Адрес +->¦ Сегмент стека  ¦ ¦
    + - - - +  ¦- - - - - - - - ¦ ¦ Перемещаемые
 DS ¦ Адрес +->¦ Сегмент данных ¦ ¦
    + - - - +  ¦- - - - - - - - ¦ ¦ в памяти
 CS ¦ Адрес +->¦ Сегмент кода   ¦ ¦
    L--------  ¦- - - - - - - - ¦ -
  Сегментные  ¦                 ¦
   регистры   ¦                 ¦
              L================-
                    Память
```

Рис.1.2. Сегменты и регистры.

Внутри программы все адреса памяти относительны к началу сегмента. Такие адреса называются смещением от начала сегмента. Двухбайтовое смещение (16-бит) может быть в пределах от шест.0000 до шест.FFFF или от 0 до 65535. Для обращения к любому адресу в программе, компьютер складывает адрес в регистре сегмента и смещение. Например, первый байт в сегменте кодов имеет смещение 0, второй байт - 01 и так далее до смещения 65535.

В качестве примера адресации, допустим, что регистр сегмента данных содержит шест.045F и некоторая команда обращается к ячейке памяти внутри сегмента данных со смещением 0032. Несмотря на то, что регистр сегмента данных содержит 045F, он указывает на адрес 045F0, т.е. на границе параграфа. Действительный адрес памяти поэтому будет следующий:

```
Адрес в DS:        045F0
Смещение:            0032
                   -----
Реальный адрес:     04622
```

Каким образом процессоры 8086/8088 адресуют память в один миллион байт? В регистре содержится 16 бит. Так как адрес сегмента всегда на границе параграфа, младшие четыре бита адреса равны нулю. Шест.FFF0 позволяет адресовать до 65520 (плюс смещение) байт. Но специалисты решили, что нет смысла иметь место для битов, которые всегда равны нулю. Поэтому адрес хранится в сегментном регистре как шест. nnnn, а компьютер полагает, что имеются еще четыре нулевых младших бита (одна шест. цифра), т.е. шест. nnnn0. Таким образом, шест.FFFF0 позволяет адресовать до 1048560 байт. Если вы сомневаетесь, то декодируйте каждое шест.F как двоичное 1111, учтите нулевые биты и сложите значения для единичных бит.

Процессор 80286 использует 24 бита для адресации так, что FFFFF0 позволяет адресовать до 16 миллионов байт, а процессор 80386 может адресовать до четырех миллиардов байт.

РЕГИСТРЫ

Процессоры 8086/8088 имеют 14 регистров, используемых для управления выполняющейся программой, для адресации памяти и для обеспечения арифметических вычислений. Каждый регистр имеет длину в одно слово (16 бит) и адресуется по имени. Биты регистра принято нумеровать слева направо:

```
        15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

Процессоры 80286 и 80386 имеют ряд дополнительных регистров, некоторые из них 16-битовые. Эти регистры здесь не рассматриваются.

Сегментные регистры CS, DS, SS и ES

Каждый сегментный регистр обеспечивает адресацию 64К памяти, которая называется текущим сегментом. Как показано ранее, сегмент выравнен на границу параграфа и его адрес в сегментном регистре предполагает наличие справа четырех нулевых битов.

1. Р е г и с т р CS. Регистр сегмента кода содержит начальный адрес сегмента кода. Этот адрес плюс величина смещения в командном указателе (IP) определяет адрес команды, которая должна быть выбрана для выполнения. Для обычных программ нет необходимости делать ссылки на регистр CS.
2. Р е г и с т р DS. Регистр сегмента данных содержит начальный адрес сегмента данных. Этот адрес плюс величина смещения, определенная в команде, указывают на конкретную ячейку в сегменте данных.
3. Р е г и с т р SS. Регистр сегмента стека содержит начальный адрес в сегменте стека.
4. Р е г и с т р ES. Некоторые операции над строками используют дополнительный сегментный регистр для управления адресацией памяти. В данном контексте регистр ES связан с индексным регистром DI. Если необходимо использовать регистр ES, ассемблерная программа должна его инициализировать.

Регистры общего назначения: AX, BX, CX и DX

При программировании на ассемблере регистры общего назначения являются "рабочими лошадками". Особенность этих регистров состоит в том, что возможна адресация их как одного целого слова или как однобайтовой части. Левый байт является старшей частью (high), а правый - младшей частью (low). Например, двухбайтовый регистр CX состоит из двух однобайтовых: CH и CL, и ссылки на регистр возможны по любому из этих трех имен. Следующие три ассемблерные команды засылают нули в регистры CX, CH и CL, соответственно:

```
MOV  CX,00
MOV  CH,00
MOV  CL,00
```

1. Р е г и с т р AX. Регистр AX является основным сумматором и применяется для всех операций ввода-вывода, некоторых операций над строками и некоторых арифметических операций. Например, команды умножения, деления и сдвига предполагают использование регистра AX. Некоторые команды генерируют более эффективный код, если они имеют ссылки на регистр AX.

   AX: | AH | AL |

2. Р е г и с т р BX. Регистр BX является базовым регистром. Это единственный регистр общего назначения, который может использоваться в качестве "индекса" для расширенной адресации. Другое общее применение его - вычисления.

   BX: | BH | BL |

3. Р е г и с т р CX. Регистр CX является счетчиком. Он необходим для управления числом повторений циклов и для операций сдвига влево или вправо. Регистр CX используется также для вычислений.

CX: | CH | CL |

4. Р е г и с т р DX. Регистр DX является регистром данных. Он применяется для некоторых операций ввода/вывода и тех операций умножения и деления над большими числами, которые используют регистровую пару DX и AX.

DX: | DH | DL |

Любые регистры общего назначения могут использоваться для сложения и вычитания как 8-ми, так и 16-ти битовых значений.

Регистровые указатели: SP и BP

Регистровые указатели SP и BP обеспечивают системе доступ к данным в сегменте стека. Реже они используются для операций сложения и вычитания.

1. Р е г и с т р SP. Указатель стека обеспечивает использование стека в памяти, позволяет временно хранить адреса и иногда данные. Этот регистр связан с регистром SS для адресации стека.
2. Р е г и с т р BP. Указатель базы облегчает доступ к параметрам: данным и адресам переданным через стек.

Индексные регистры: SI и DI

Оба индексных регистра возможны для расширенной адресации и для использования в операциях сложения и вычитания.

1. Р е г и с т р SI. Этот регистр является индексом источника и применяется для некоторых операций над строками. В данном контексте регистр SI связан с регистром DS.
2. Р е г и с т р DI. Этот регистр является индексом назначения и применяется также для строковых операций. В данном контексте регистр DI связан с регистром ES.

Регистр командного указателя: IP

Регистр IP содержит смещение на команду, которая должна быть выполнена. Обычно этот регистр в программе не используется, но он может изменять свое значение при использовании отладчика DOS DEBUG для тестирования программы.

Флаговый регистр

Девять из 16 битов флагового регистра являются активными и определяют текущее состояние машины и результатов выполнения. Многие арифметические команды и команды сравнения изменяют состояние флагов. Назначение флаговых битов:

Флаг	Назначение
O (Переполнение)	Указывает на переполнение старшего бита при арифметических командах.
D (Направление)	Обозначает левое или правое направление пересылки или сравнения строковых данных (данных в памяти превышающих длину одного слова).

I (Прерывание)   Указывает на возможность внешних прерываний.
T (Пошаговый режим)   Обеспечивает возможность работы процессора в пошаговом режиме. Например, программа DOS DEBUG устанавливает данный флаг так, что возможно пошаговое выполнение каждой команды для проверки изменения содержимого регистров и памяти.
S (Знак)   Содержит результирующий знак после арифметических операций (0 - плюс, 1 - минус).
Z (Ноль)   Показывает результат арифметических операций и операций сравнения (0 - ненулевой, 1 - нулевой результат).
A (Внешний перенос)   Содержит перенос из 3-го бита для 8-битных данных используется для специальных арифметических операций.
P (Контроль четности)   Показывает четность младших 8-битовых данных (1 - четное и 0 - нечетное число).
C (Перенос)   Содержит перенос из старшего бита, после арифметических операций, а также последний бит при сдвигах или циклических сдвигах.

При программировании на ассемблере наиболее часто используются флаги O, S, Z, и C для арифметических операций и операций сравнения, а флаг D для обозначения направления в операциях над строками. В последующих главах содержится более подробная информация о флаговом регистре.

АРХИТЕКТУРА PC

Основными элементами аппаратных средств компьютера являются: системный блок, клавиатура, устройство отображения, дисководы, печатающее устройство (принтер) и различные средства для асинхронной коммуникации и управления игровыми программами. Системный блок состоит из системной платы, блока питания и ячейки расширения для дополнительных плат. На системной плате размещены:

- микропроцессор (Intel);
- постоянная память (ROM 40Кбайт);
- оперативная память (RAM до 512К в зависимости от модели);
- расширенная версия бейсик-интерпретатора.

Ячейки расширения обеспечивают подключение устройств отображения, дисководов для гибких дисков (дискет), каналов телекоммуникаций, дополнительной памяти и игровых устройств.

Клавиатура содержит собственный микропроцессор, который обеспечивает тестирование при включении памяти, сканирование клавиатуры, подавление "дребезга" клавишей и буферизацию до 20 символов.

"Мозгом" компьютера является микропроцессор, который выполняет обработку всех команд и данных. Процессор 8088 использует 16-битовые регистры, которые могут обрабатывать два байта одновременно. Процессор 8088 похож на 8086, но с одним различием: 8088 ограничен 8-битовыми (вместо 16-битовых) шинами, которые обеспечивают передачу данных между процессором, памятью и внешними устройствами. Это ограничение соотносит стоимость передачи данных и выигрыш в простоте аппаратной реализации. Процессоры 80286 и 80386 являются расширенными версиями процессора 8086.

Как показано на рис.1.3 процессор разделен на две части: операционное устройство (ОУ) и шинный интерфейс (ШИ). Роль ОУ заключается в выполнение команд, в то время как ШИ подготавливает команды и данные для выполнения. Операционное устройство содержит арифметико-логическое устройство (АЛУ), устройство управления (УУ) и десять регистров. Эти устройства обеспечивают выполнение команд, арифметические вычисления и логические операции

(сравнение на больше, меньше или равно).

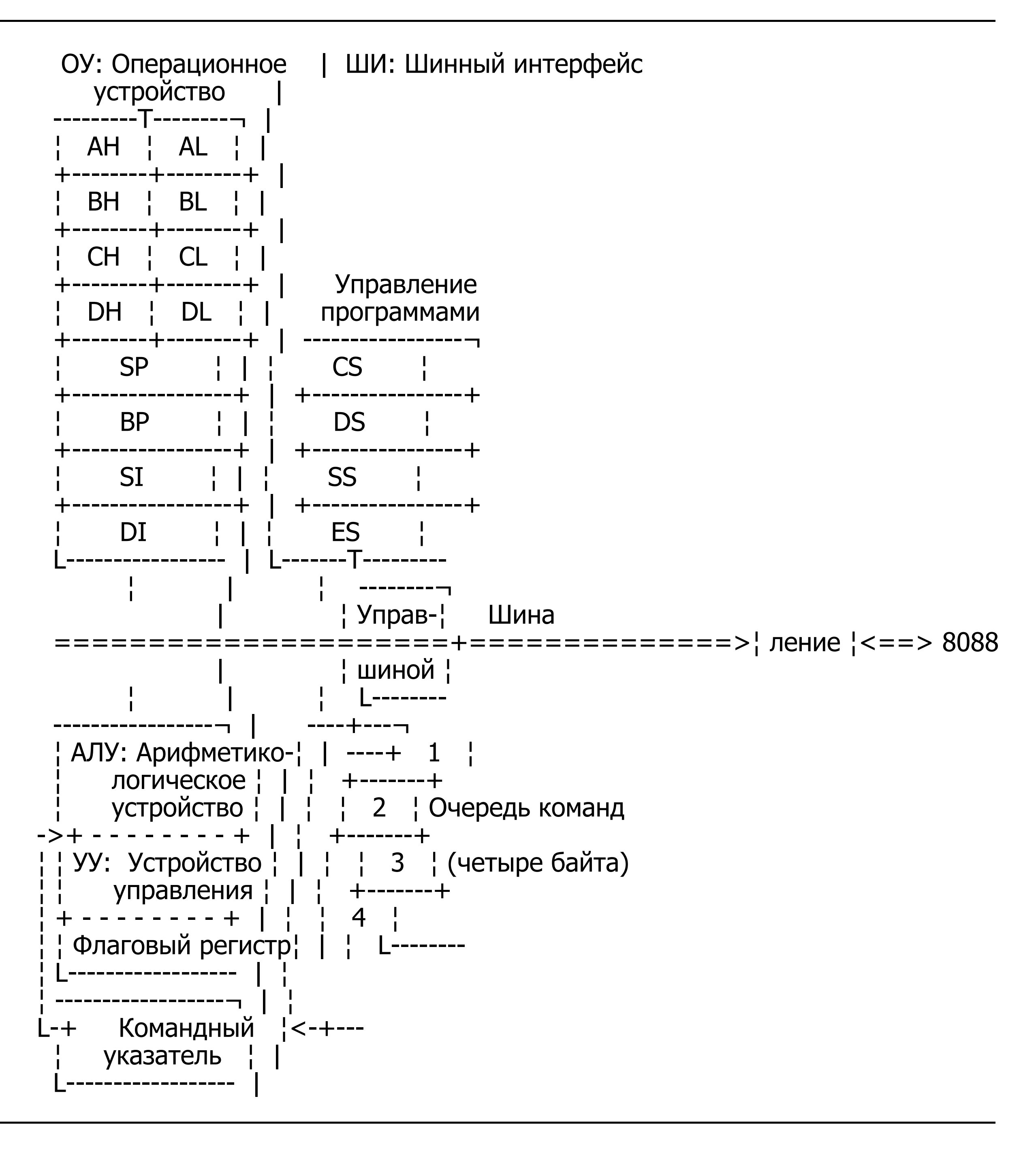


Рис.1.3. Операционное устройство и шинный интерфейс.

Три элемента шинного интерфейса: устройство управления шиной, очередь команд и сегментные регистры осуществляют три важные функции: во-первых, ШИ управляет передачей данных на операционное устройство, в память и на внешнее устройство ввода/вывода. Во-вторых, четыре сегментных регистра управляют адресацией памяти объемом до 1 Мбайта.

Третья функция ШИ это выборка команд. Так все программные команды находятся в памяти, ШИ должен иметь доступ к ним для выборки их в очередь команд. Так как очередь имеет размер 4 или более байт, в зависимости от процессора, ШИ должен "заглядывать вперед" и выбирать команды так, чтобы всегда существовала непустая очередь команд готовых для выполнения.

Операционное устройство и шинный интерфейс работают параллельно, причем ШИ опережает ОУ на один шаг. Операционное устройство сообщает шинному интерфейсу о необходимости доступа к данным в памяти или на устройство ввода/вывода. Кроме того ОУ запрашивает машинные команды из

очереди команд. Пока ОУ занято выполнением первой в очереди команды, ШИ выбирает следующую команду из памяти. Эта выборка происходит во время выполнения, что повышает скорость обработки.

Память

Обычно микрокомпьютер имеет два типа внутренней памяти. первый тип это постоянная память (ПЗУ) или ROM (read-only memory). ROM представляет собой специальную микросхему, из которой (как это следует из названия) возможно только чтение. Поскольку данные в ROM специальным образом "прожигаются" они не могут быть модифицированы.

Основным назначением ROM является поддержка процедур начальной загрузки: при включении питания компьютера ROM выполняет различные проверки и загружает в оперативную память (RAM) данные из системной дискеты (например, DOS). Для целей программирования наиболее важным элементом ROM является BIOS (Basic Input/Output System) базовая система ввода/вывода, которая рассматривается в следующих главах. (Basic - здесь обычное слово, а не язык программирования). ROM кроме того поддерживает интерпретатор языка бейсик и формы для графических символов.

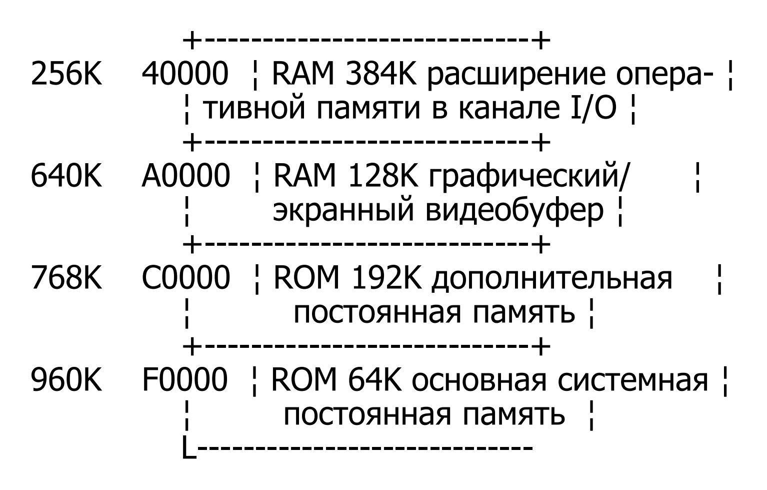
Память, с которой имеет дело программист, представляет собой RAM (Random Access Memory) или ОЗУ, т.е. оперативная памяти, доступная как для чтения, так и для записи. RAM можно рассматривать как рабочую область для временного хранения программ и данных на время выполнения.

Так как содержимое RAM теряется при отключении питания компьютера, необходима внешняя память для сохранения программ и данных. Если установлена дискета с операционной системой или имеется жесткий диск типа винчестер, то при включении питания ROM загружает программы DOS в RAM. (Загружается только основная часть DOS, а не полный набор программ DOS). Затем необходимо ответить на приглашение DOS для установки даты и можно вводить запросы DOS для выполнения конкретных действий. Одним из таких действий может быть загрузка программ с диска в RAM. Поскольку DOS не занимает всю память, то в ней имеется (обычно) место для пользовательских программ. Пользовательская программа выполняется в RAM и обычно осуществляет вывод на экран, принтер или диск. По окончании можно загрузить другую программу в RAM. Предыдущая программа хранится на диске и новая программа при загрузке может наложиться (затереть) предыдущую программу в RAM.

В ы д е л е н и е п а м я т и. Так как любой сегмент имеет объем до 64К и имеется четыре типа сегментов, то это предполагает общее количество доступной RAM памяти: 4 x 64К = 256К. Но возможно любое количество сегментов. Для того, чтобы адресовать другой сегмент, необходимо всего лишь изменить адрес сегментного регистра.

RAM включает в себя первые три четверти памяти, а ROM - последнюю четверть. В соответствии с картой физической памяти микрокомпьютера, приведенной на рис.1.4, первые 256К RAM памяти находятся на системной плате. Так как одна область в RAM зарезервирована для видеобуфера, то имеется 640К доступных для использования программистом, по крайней мере в текущих версиях DOS. ROM начинается по адресу 768К и обеспечивает поддержку операций ввода/вывода на такие устройства как контролер жесткого диска. ROM, начинающийся по адресу 960К управляет базовыми функциями компьютера, такими как тест при включении питания, точечные образы графических символов и автозагрузчик с дискет.

```
   Начальный адрес         Память
   Дес.   Шест.  -----------------------------¬
   0K     00000 ¦ RAM 256K основная           ¦
                ¦      оперативная память ¦
```



```
            +----------------------------+
256K    40000  ¦ RAM 384K расширение опера- ¦
               ¦ тивной памяти в канале I/O ¦
               +----------------------------+
640K    A0000  ¦ RAM 128K графический/      ¦
               ¦      экранный видеобуфер ¦
               +----------------------------+
768K    C0000  ¦ ROM 192K дополнительная    ¦
               ¦        постоянная память ¦
               +----------------------------+
960K    F0000  ¦ ROM 64K основная системная ¦
               ¦       постоянная память  ¦
               L-----------------------------
```

Рис.1.4. Карта физической памяти.

Все дальнейшие упоминания RAM используют общий термин - память.

А д р е с а ц и я. Все ячейки памяти пронумерованы последовательно от 00 - минимального адреса памяти. Процессор обеспечивает доступ к байтам или словам в памяти. Рассмотрим десятичное число 1025. Для записи в память шест. представления этого числа - 0401 требуется два байта или одно слово. Оно состоит из старшей части - 04 и младшей части - 01. Система хранит в памяти байты слова в обратной последовательности: младшая часть по меньшему адресу, а старшая - по большему адресу. Предположим, что процессор записал шест.0401 из регистра в ячейки памяти 5612 и 5613, следующим образом:

```
          |01|04|
           |  |
    ячейка 5612,  ячейка 5613
    младший байт  старший байт
```

Процессор полагает, что байты числовых данных в памяти представлены в обратной последовательности и обрабатывает их соответственно. Несмотря на то, что это свойство полностью автоматизировано, следует всегда помнить об этом факте при программировании и отладке ассемблерных программ.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Единицей памяти является байт, состоящий из восьми информационных и одного контрольного битов. Два смежных байта образуют слово.

- Сердцем компьютера является микропроцессор, который имеет доступ к байтам или словам в памяти.

- ASCII код есть формат представлением символьных данных.

- Компьютер способен различать биты, имеющие разное значение: 0 или 1, и выполнять арифметические операции только в двоичном формате.

- Значение двоичного числа определено расположением единичных битов. Так, двоичное 1111 равно 2**3 + 2**2 + 2**1 + 2**0, или 15.

- Отрицательные числа представляются двоичным дополнением: обратные значения бит положительного представления числа +1.

- Сокращенная запись групп из четыре битов представляет собой шестнадцатиричный формат. Шест. цифры 0-9 и A-F представляют двоичные числа от 0000 до 1111.

- Программы состоят из сегментов: сегмент стека для хранения адресов возврата, сегмент данных для определения данных и рабочих областей и сегмент кода для выполняемых команд. Все адреса в программе представлены как относительные смещения от начала сегмента.

- Регистры управляют выполнением команд, адресацией, арифметическими операциями и состоянием выполнения.

- ROM (ПЗУ) и RAM (ОЗУ) представляют собой два типа внутренней памяти.

- Процессор хранит двухбайтовые числовые данные (слова) в памяти в обратной последовательности.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1.1. Напишите битовые представления ASCII кодов для следующих однобитовых символов. (Используйте приложение 1 в качестве справочника): а) P, б) p, в) #, г) 5.

1.2. Напишите битовые представления для следующих чисел: а) 5, б) 13, в) 21, г) 27.

1.3. Сложите следующие двоичные:

```
а) 00010101    б) 00111110    в) 00011111
   00001101       00101001       00000001
   --------       --------       --------
```

1.4. Определите двоичные дополнения для следующих двоичных чисел: а) 00010011, б) 00111100, в) 00111001.

1.5. Определите положительные значения для следующих отрицательных двоичных чисел: а) 11001000, б) 10111101, в) 10000000.

1.6. Определите шест. представления для а) ASCII символа Q, б) ASCII числа 7, в) двоичного числа 01011101, г) двоичного 01110111.

1.7. Сложите следующие шест. числа:

```
а) 23A6   б) 51FD   в) 7779   г) EABE
   0022        3        887      26C4
   ----      ----      ----      ----
```

1.8. Определите шест. представления для следующих десятичных чисел. Метод преобразования приведен в прил.2. Проверьте также полученные результаты, преобразовав шест. значения в двоичные и сложив единичные биты: а) 19, б) 33, в) 89, г) 255, д) 4095, е) 63398.

1.9. Что представляют собой три типа сегментов, каковы их максимальные размеры и адреса, с которых они начинаются.

1.10. Какие регистры можно использовать для следующих целей: а) сложение и вычитание, б) подсчет числа циклов, в) умножение и деление, г) адресация сегментов, д) индикация нулевого результата, е) адресация

выполняемой команды?

1.11. Что представляют собой два основных типа памяти компьютера и каково их основное назначение?

ГЛАВА 2 Выполнение программ

Ц е л ь: Представить машинный язык, ввод команд в память и выполнение программ.

ВВЕДЕНИЕ

Основой данной главы является использование DOS программы с именем DEBUG, которая позволяет просматривать память, вводить программы и осуществлять трассировку их выполнения. В главе показан процесс ввода этих программ непосредственно в память в область сегмента кодов и объяснен каждый шаг выполнения программы.

Начальные упражнения научат проверять содержимое конкретных ячеек памяти. В первом примере программы используются непосредственные данные определенные в командах загрузки регистров и арифметических командах. Второй пример программы использует данные, определенные отдельно в сегменте данных. Трассировка этих команд в процессе выполнения программы позволяет понять действия компьютера и роль регистров.

Для начала не требуется предварительных знаний языка ассемблера и даже программирования. Все что необходимо - это IBM PC или совместимый микрокомпьютер и диск с операционной системой DOS.

НАЧАЛО РАБОТЫ

Прежде всего необходимо вставить дискету с DOS в левый дисковод A. Если питание выключено, то его надо включить; если питание уже включено, нажмите вместе и задержите клавиши Ctrl и Alt и нажмите клавишу Del.

Когда рабочая часть DOS будет загружена в память, на экране появится запрос для ввода даты и времени, а затем буква текущего дисковода, обычно A для дискеты и C для винчестера (твердого диска). Изменить текущий дисковод можно, нажав соответствующую букву, двоеточие и клавишу Return. Это обычная процедура загрузки, которую следует использовать всякий раз для упражнений из этой книги.

ПРОСМОТР ЯЧЕЕК ПАМЯТИ

В этом первом упражнении для просмотра содержимого ячеек памяти используется программа DOS DEBUG. Для запуска этой пограммы введите DEBUG и нажмите Return, в результате программа DEBUG должна загрузится с диска в память. После окончания загрузки на экране появится приглашение в виде дефиса, что свидетельствует о готовности программы DEBUG для приема команд. Единственная команда, которая имеет отношение к данному упражнению, это D - для дампа памяти.

1. Р а з м е р п а м я т и. Сначала проверим размер доступной для работы памяти. В зависимости от модели компьютера это значение связано с установкой внутренних переключателей и может быть меньше,

чем реально существует. Данное значение находится в ячейках памяти шест.413 и 414 и его можно просмотреть из DEBUG по адресу, состоящему из двух частей:

400 - это адрес сегмента, который записывается как 40 (последний нуль подразумевается) и
13 - это смещение от начала сегмента. Таким образом, можно ввести следующий запрос:

D 40:13 (и нажать Return)

Первые два байта, появившиеся в результате на экране, содержат размер памяти в килобайтах и в шестнадцатеричном представлении, причем байты располагаются в обратной последовательности. Несколько следующих примеров показывают шест. обратное, шест. нормальное и десятичные представления.

Шест. обратн.	Шест. норм.	Десятичн. (K)
8000	0080	128
0001	0100	256
8001	0180	384
0002	0200	512
8002	0280	640

2. С е р и й н ы й н о м е р. Серийный номер компьютера "зашит" в ROM по адресу шест. FE000. Чтобы увидеть его, следует ввести:

D FE00:0 (и нажать Return)

В результате на экране появится семизначный номер компьютера и дата копирайт.
3. Д а т а ROM BIOS. Дата ROM BIOS в формате mm/dd/yy находится по шест. адресу FFFF5. Введите

D FFFF:05 (и нажмите Return)

знание этой информации (даты) иногда бывает полезным для определения модели и возраста компьютера.

Теперь, поскольку вы знаете, как пользоваться командой D (Display), можно устанавливать адрес любой ячейки памяти для просмотра содержимого. Можно также пролистывать память, периодически нажимая клавишу D, - DEBUG выведет на экран адреса, следующие за последней командой.
Для окончания работы и выхода из отладчика в DOS введите команду Q (Quit). Рассмотрим теперь использование отладчика DEBUG для непосредственного ввода программ в память и трассировки их выполнения.

ПРИМЕР МАШИННЫХ КОДОВ: НЕПОСРЕДСТВЕННЫЕ ДАННЫЕ

Цель данного примера - проиллюстрировать простую программу на машинном языке, ее представление в памяти и результаты ее выполнения. Программа показана в шестнадцатиричном формате:

Команда	Назначение
B82301	Переслать шест. значение 0123 в AX.
052500	Прибавить шест. значение 0025 к AX.
8BD8	Переслать содержимое AX в BX.
03D8	Прибавить содержимое AX к BX.

8BCB   Переслать содержимое BX в CX.
2BC8   Вычесть содержимое AX из AX (очистка AX).
90   Нет операции.
CB   Возврат в DOS.

Можно заметить, что машинные команды имеют различную длину: один, два или три байта. Машинные команды находятся в памяти непосредственно друг за другом. Выполнение программы начинается с первой команды и далее последовательно выполняются остальные. Не следует, однако, в данный момент искать большой смысл в приведенном машинном коде. Например, в одном случае MOV - шест.B8, а в другом - шест.8B.

Можно ввести эту программу непосредственно в память машины и выполнить ее покомандно. В тоже время можно просматривать содержимое регистров после выполнения каждой команды. Начнем данное упражнение так же как делалось предыдущее - ввод команды отладчика DEBUG и нажатие клавиши Return. После загрузки DEBUG на экране высвечивается приглашение к вводу команд в виде дефиса. Для печати данного упражнения включите принтер и нажмите Ctrl и PrtSc одновременно.

Для непосредственного ввода программы на машинном языке введите следующую команду, включая пробелы:

E CS:100 B8 23 01 05 25 00 (нажмите Return)

Команда E обозначает Enter (ввод). CS:100 определяет адрес памяти, куда будут вводиться команды, - шест.100 (256) байт от начала сегмента кодов. (Обычный стартовый адрес для машинных кодов в отладчике DEBUG). Команда E записывает каждую пару шестнадцатиричных цифр в память в виде байта, начиная с адреса CS:100 до адреса CS:105.

Следующая команда Enter:

E CS:106 8B D8 03 D8 8B CB (Return)

вводит шесть байтов в ячейки, начиная с адреса CS:106 и далее в 107, 108, 109, 10A и 10B. Последняя команда Enter:

E CS:10C 2B C8 2B C0 90 CB (Return)

вводит шесть байтов, начиная с CS:10C в 10D, 10E, 10F, 110 и 111. Проверьте правильность ввода значений. Если есть ошибки, то следует повторить команды, которые были введены неправильно.

Теперь осталось самое простое - выполнить эти команды. На рис.2.1 показаны все шаги, включая команды E. На вашем экране должны быть аналогичные результаты после ввода каждой команды отладчика.

Введите команду R для просмотра содержимого регистров и флагов. В данный момент отладчик покажет содержимое регистров в шест. формате, например,

AX=0000, BX=0000, ...

В зависимости от версии DOS содержимое регистров на экране может отличаться от показанного на рис.2.1. Содержимое регистра IP (указатель команд) выводится в виде IP=0100, показывая что выполняемая команда находится на смещении 100 байт от начала сегмента кодов. (Вот почему использовалась команда E CS:100 для установки начала программы.)

Регистр флагов на рис.2.1 показывает следующие значения флагов:

NV UP DI PL NZ NA PO NC

Данные значения соответствуют: нет переполнения, правое направление, прерывания запрещены, знак плюс, не ноль, нет внешнего переноса, контроль на честность и нет переноса. В данный момент значение флагов не

существенно.

```
D>DEBUG
-E CS:100 B8 23 01 05 25 00
-E CS:106 8B D8 03 D8 8B CB
-E CS:10C 2B C8 2B C0 90 CB
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0100   NV UP EI PL NZ NA PO NC
13C6:0100 B8230         MOV     AX,0123
-T

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0103   NV UP EI PL NZ NA PO NC
13C6:0103 052500        ADD     AX,0025
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0106   NV UP EI PL NZ NA PE NC
13C6:0106 8BD8          MOV     BX,AX
-T

AX=0148  BX=0148  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0108   NV UP EI PL NZ NA PO NC
13C6:0108 03D8          ADD     BX,AX
-T

AX=0148  BX=0290  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=010A   NV UP EI PL NZ AC PO NC
13C6:010A 8BCB          MOV     CX,BX
-T

AX=0148  BX=0290  CX=0290  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=010C   NV UP EI PL NZ AC PO NC
13C6:010C 2BC8          SUB     CX,AX

-T

AX=0148  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0100   NV UP EI PL NZ AC PO NC
13C6:010E 2BC0          SUB     AX,AX
-T

AX=0000  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0110   NV UP EI PL ZR NA PO NC
13C6:0110 90            NOP
-T

AX=0000  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0111   NV UP EI PL ZR NA PO NC
13C6:0111 CB            RETF
-
```

Рис.2.1. Трассировка машинных команд.

Команда R показывает также по смешению 0100 первую выполняемую машинную команду. Регистр CS на рис.2.1 содержит значение CS=13C6 (на

разных компьютерах оно может различаться), а машинная команда выглядит следующим образом:

```
13C6:0100  B82301   MOV  AX,0123
```

CS=13C6 обозначает, что начало сегментов кода находится по смещению 13C6 или точнее 13C60. Значение 13C6:0100 обозначает 100 (шест.) байтов от начального адреса 13C6 в регистре CS.

B82301 - машинная команда, введенная по адресу CS:100.

MOV AX,0123 - ассемблерный мнемонический код, соответствующий введенной машинной команде. Это есть результат операции дисассемблирования, которую обеспечивает отладчик для более простого понимания машинных команд. В последующих главах мы будем кодировать программы исключительно в командах ассемблера. Рассматриваемая в данном случае команда обозначает пересылку непосредственного значения в регистр AX.

В данный момент команда MOV еще не выполнена. Для ее выполнения нажмите клавишу T (для трассировки) и клавишу Return. В результате команда MOV будет выполнена и отладчик выдаст на экран содержимое регистров, флаги, а также следующую на очереди команду. Заметим, что регистр AX теперь содержит 0123. Машинная команда пересылки в регистр AX имеет код B8 и за этим кодом следует непосредственные данные 2301. В ходе выполнения команда B8 пересылает значение 23 в младшую часть регистра AX, т.е. однобайтовый регистр AL, а значение 01 - в старшую часть регистра AX, т.е. в регистр AH:

```
AX: | 01 | 23 |
```

Содержимое регистра IP:0103 показывает адрес следующей выполняемой команды в сегменте кодов:

```
13C6:0103  052500   ADD  AX,0025
```

Для выполнения данной команды снова введите T. Команда прибавит 25 к младшей (AL) части регистра AX и 00 к старшей (AH) части регистра AX, т.е. прибавит 0025 к регистру AX. Теперь регистр AX содержит 0148, а регистр IP 0106 - адрес следующей команды для выполнения.

Введите снова команду T. Следующая машинная команда пересылает содержимое регистра AX в регистр BX и после ее выполнения в регистре BX будет содержаться значение 0148. Регистр AX сохраняет прежнее значение 0148, поскольку команда MOV только копирует данные из одного места в другое.

Теперь вводите команду T для пошагового выполнения каждой оставшейся в программе команды. Следующая команда прибавит содержимое регистра AX к содержимому регистра BX, в последнем получим 0290. Затем программа скопирует содержимое регистра BX в CX, вычтет AX из CX, и вычтет AX из него самого. После этой последней команды, флаг нуля изменит свое состояние с NZ (не нуль) на ZR (нуль), так как результатом этой команды является нуль (вычитание AX из самого себя очищает этот регистр в 0).

Можно ввести T для выполнения последних команд NOP и RET, но это мы сделаем позже. Для просмотра программы в машинных кодах в сегменте кодов введите D для дампа:

```
D CS:100
```

В результате отладчик выдаст на каждую строку экрана по 16 байт данных в шест. представлении (32 шест. цифры) и в символьном представлении в коде ASCII (один символ на каждую пару шест. цифр). Представление машинного кода в символах ASCII не имеет смысла и может быть игнорировано. В следующих разделах будет рассмотрен символьный дамп более подробно.

Первая строка дампа начинается с 00 и представляет содержимое ячеек от CS:100 до CS:10F. Вторая строка представляет содержимое ячеек от CS:110 до CS:11F. Несмотря на то, что ваша программа заканчивается по адресу CS:111, команда Dump автоматически выдаст на восьми строках экрана дамп с адреса CS:100 до адреса CS:170.

При необходимости повторить выполнение этих команд сбросьте содержимое регистра IP и повторите трассировку снова. Введите R IP, введите 100, а затем необходимое число команд T. После каждой команды нажимайте клавишу Return.

На рис.2.2 показан результат выполнения команды D CS:100. Обратите внимание на машинный код с CS:100 до 111 и вы обнаружите дамп вашей программы; следующие байты могут содержать любые данные.

```
-D CS:100
13C6:0100 B8 23 01 05 25 00 8B D8-03 D8 8B CB 2B C8 2B C0 .#..%.......+.+.
13C6:0110 90 CB 8D 46 14 50 51 52-FF 76 28 E8 74 00 8B E5 ...F.PQR.v(.t...
13C6:0120 B8 01 00 50 FF 76 32 FF-76 30 FF 76 2E FF 76 28 ...P.v2.v0.v..v(
13C6:0130 E8 88 15 8B E5 BF 36 18-12 FF 36 16 12 8B 76 28 ......6...6...v(
13C6:0140 FF 74 3A 89 46 06 E8 22-CE 8B E5 30 E4 3D 0A 00 .t:.F.."...0.=..
13C6:0150 75 32 A1 16 12 2D 01 00-8B 1E 18 12 83 DB 00 53 u2...-.........S
13C6:0160 50 8B 76 28 FF 74 3A A3-16 12 89 1E 18 12 E8 FA P.v(.t:.........
13C6:0170 CD 8B E5 30 E4 3D 0D 00-74 0A 83 06 16 12 01 83 ...0.=..t.......
-Q
```

Рис.2.2. Дамп кодового сегмента.

Для завершения работы с программой DEBUG введите Q (Quit - выход). В результате произойдет возврат в DOS и на экране появится приглашение A> или C>. Если печатался протокол работы с отладчиком, то для прекращения печати снова нажмите Ctrl/PrtSc.

ПРИМЕР МАШИННЫХ КОДОВ: ОПРЕДЕЛЕНИЕ ДАННЫХ

В предыдущем примере использовались непосредственные данные, описанные непосредственно в первых двух командах (MOV и ADD). Теперь рассмотрим аналогичный пример, в котором значения 0123 и 0025 определены в двух полях сигмента данных. Данный пример позволяет понять как компьютер обеспечивает доступ к данным посредством регистра DS и адресного смещения.

В настоящем примере определены области данных, содержащие соответственно следующие значения:

Адрес в DS	Шест. знач.	Номера байтов
0000	2301	0 и 1
0002	2500	2 и 3
0004	0000	4 и 5
0006	2A2A2A	6, 7 и 8

Вспомним, что шест. символ занимает половину байта, таким образом, например, 23 находится в байте 0 (в первом байте) сегмента данных, 01 - в байте 1 (т.е. во втором байте).

Ниже показаны команды машинного языка, которые обрабатывают эти данные:

Команда Назначение

A10000 Переслать слово (два байта), начинающееся в DS по адресу 0000, в регистр AX.
03060200 Прибавить содержимое слова (двух байт), начинающегося в DS по адресу 0002, к регистру AX.
A30400 Переслать содержимое регистра AX в слово, начинающееся в DS по адресу 0004.
CB Вернуться в DOS.

Обратите внимание, что здесь имеются две команды MOV с различными машинными кодами: A1 и A3. Фактически машинный код зависит от регистров, на которые имеется ссылка, количества байтов (байт или слово), направления передачи данных (из регистра или в регистр) и от ссылки на непосредственные данные или на память.

Воспользуемся опять отладчиком DEBUG для ввода данной программы и анализа ее выполнения. Когда отладчик выдал свое дефисное приглашение, он готов к приему команд.

Сначала введите команды E (Enter) для сегмента данных:

```
E DS:00 23 01 25 00 00 00 (Нажмите Return)
E DS:06 2A 2A 2A (Нажмите Return)
```

Первая команда записывает три слова (шесть байтов) в начало сегмента данных, DS:00. Заметьте, что каждое слово вводилось в обратной последовательности, так что 0123 есть 2301, а 0025 есть 2500. Когда команда MOV будет обращаться к этим словам, нормальная последовательность будет восстановлена и 2301 станет 0123, а 2500 - 0025.

Вторая команда записывает три звездочки (***) для того, чтобы их можно было видеть впоследствии по команде D (Dump) - другого назначения эти звездочки не имеют.

Введем теперь команды в сегмент кодов, опять начиная с адреса CS:100:

```
E CS:100 A1 00 00 03 06 02 00
E CS:107 A3 04 00 CB
```

Теперь команды находятся в ячейках памяти от CS:100 до CS:10A. Эти команды можно выполнить как это делалось ранее. На рис.2.3 показаны все шаги, включая команды E. На экране дисплея должны появиться такие же результаты, хотя адреса CS и DS могут различаться. Для пересмотра информации в сегменте данных и в сегменте кодов введите команды D (Dump) соответственно:

```
для сегмента данных:    D DS:000 (Return)
для сегмента кодов:     D CS:100 (Return)
```

Сравните содержимое обоих сегментов с тем, что вводилось и с изображенным на рис.2.3. Содержимое памяти от DS:00 до DS:08 и от CS:100 до CS:10A должно быть идентично рис.2.3.

```
D>DEBUG
-E DS:23 01 25 00 00
-E DS:2A 2A 2A
-E CS:100 A1 00 00 03 06 02 00
-E CS:107 A3 04 00 CB
-D DS:0
13C6:0000 23 01 25 00 00 9A 2A 2A-2A F0 F5 02 2C 10 2E 03 #.%...***...,...
13C6:0010 2C 10 BD 02 2C 10 B1 0D-01 03 01 00 02 FF FF FF ,...,...........
13C6:0020 FF FF FF FF FF FF FF FF-FF FF FF FF EF 0F 64 00 ..............d.
13C6:0030 61 13 14 00 18 00 C7 13-FF FF FF FF 00 00 00 00 a..............
```

```
13C6:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
13C6:0050 CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20 .!...........
13C6:0060 20 20 20 20 20 20 20 20 00 00 00 00 00 20 20 20         .....
13C6:0070 20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00         ........
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0100   NV UP EI PL NZ NA PO NC
13C6:0100 A10000        MOV     AX,[0000]                DS:0000=0123
-T

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0103   NV UP EI PL NZ NA PO NC
13C6:0103 03060200      ADD     AX,[0002]                DS:0002=0025
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0107   NV UP EI PL NZ NA PE NC
13C6:0107 A30400        MOV     [0004],AX                DS:0004=9A00
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=13C6  ES=13C6  SS=13C6  CS=13C6  IP=0108   NV UP EI PL NZ NA PO NC
13C6:010A CB            RETF
-D DS:0
13C6:0000 23 01 25 00 00 9A 2A 2A-2A F0 F5 02 2C 10 2E 03 #.%...***...,...
13C6:0010 2C 10 BD 02 2C 10 B1 0D-01 03 01 00 02 FF FF FF ,...,...........
13C6:0020 FF FF FF FF FF FF FF FF-FF FF FF FF EF 0F 64 00 ..............d.
13C6:0030 61 13 14 00 18 00 C7 13-FF FF FF FF 00 00 00 00 a...............
13C6:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
13C6:0050 CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20 .!...........
13C6:0060 20 20 20 20 20 20 20 20 00 00 00 00 00 20 20 20         .....
13C6:0070 20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00         ........
-Q
```

Рис.2.3. Трассировка машинных команд.

Теперь введите R для просмотра содержимого регистров и флагов и для отображения первой команды. Регистры содержат те же значения, как при старте первого примера. Команда отобразится в виде:

```
13C6:0100      A10000    MOV  AX,[0000]
```

Так, как регистр CS содержит 13C6, то CS:100 содержит первую команду A10000. Отладчик интерпретирует эту команду как MOV и определяет ссылку к первому адресу [0000] в сегменте данных. Квадратные скобки необходимы для указания ссылки к адресу памяти, а не к непосредственным данным.

Если бы квадратных скобок не было, то команда

```
MOV AX,0000
```

обнулила бы регистр AX непосредственным значением 0000.

Теперь введем команду T. Команда MOV AX,[0000] перешлет содержимое слова, находящегося по нулевому смещению в сегменте данных, в регистр AX. Содержимое 2301 преобразуется командой в 0123 и помещается в регистр AX.

Следующую команду ADD можно выполнить, введя еще раз команду T. В результате содержимое слова в DS по смещению 0002 прибавится в регистр AX. Теперь регистр AX будет содержать сумму 0123 и 0025, т.е. 0148.

Следующая команда MOV [0004],AX выполняется опять по вводу T. Эта команда пересылает содержимое регистра AX в слово по смешению 0004. Для

просмотра изменений содержимого сегмента данных введите D DS:00. Первые девять байт будут следующими:

```
значение в сегменте данных: 23 01 25 00 48 01 2A 2A 2A
величина смещения:          00 01 02 03 04 05 06 07 08
```

Значение 0148, которое было занесено из регистра AX в сегмент данных по смещению 04 и 05, имеет обратное представление 4801. Заметьте что эти шест. значения представлены в правой части экрана их символами в коде ASCII. Например, шест.23 генерируется в символ #, а шест.25 - в символ %. Три байта с шест. значениями 2A высвечиваются в виде трех звездочек (***). Левая часть дампа показывает действительные машинные коды, которые находятся в памяти. Правая часть дампа только помогает проще локализовать символьные (срочные) данные.

Для просмотра содержимого сегмента кодов введите D DS:100 так, как показано на рис.2.3. В заключении введите Q для завершения работы с программой.

МАШИННАЯ АДРЕСАЦИЯ

Для доступа к машинной команде процессор определяет ее адрес из содержимого регистра CS плюс смещение в регистре IP. Например, предположим, что регистр CS содержит шест.04AF (действительный адрес 04AF0), а регистр IP содержит шест. 0023:

```
                CS:    04AF0
                IP:     0023
                       -----
      Адрес команды:   04B13
```

Если, например, по адресу 04B13 находится команда:

```
      A11200      MOV  AX,[0012]
      |
   Адрес 04B13
```

то в памяти по адресу 04B13 содержится первый байт команды. Процессор получает доступ к этому байту и по коду команды (A1) определяет длину команды - 3 байта.

Для доступа к данным по смещению [0012] процессор определяет адрес, исходя из содержимого регистра DS (как правило) плюс смещение в операнде команды. Если DS содержит шест.04B1 (реальный адрес 04B10), то результирующий адрес данных определяется следующим образом:

```
         DS:    04B10
      Смещение:   0012
                 -----
    Адрес данных: 04B22
```

Предположим, что по адресам 04B22 и 04B23 содержатся следующие данные:

```
      Содержимое:    24      01
                   |       |
      Адрес:      04B22   04B23
```

Процессор выбирает значение 24 из ячейки по адресу 04B22 и помещает его в регистр AL, и значение 01 по адресу 04B23 - в регистр AH. Регистр AX будет содержать в результате 0124. В процессе выборки каждого байта

команды процессор увеличивает значение регистра IP на единицу, так что к началу выполнения следующей команды в нашем примере IP будет содержать смещение 0026. Таким образом процессор теперь готов для выполнения следующей команды, которую он получает по адресу из регистра CS (04AF0) плюс текущее смещение в регистре IP (0026), т.е. 04B16.

Четная адресация

Процессор 8086, 80286 и 80386 действуют более эффективно, если в программе обеспечиваются доступ к словам, расположенным по четным адресам. В предыдущем примере процессор может сделать одну выборку слова по адресу 4B22 для загрузки его непосредственно в регистр. Но если слово начинается на нечетном адресе, процессор выполняет двойную выборку. Предположим, например, что команда должна выполнить выборку слова, начинающегося по адресу 04B23 и загрузить его в регистр AX:

```
        Содержимое памяти:  |xx|24|01|xx|
                                |
                  Адрес:  04B23
```

Сначала процессор получает доступ к байтам по адресам 4B22 и 4B23 и пересылает байт из ячейки 4B23 в регистр AL. Затем он получает доступ к байтам по адресам 4B24 и 4B25 и пересылает байт из ячейки 4B23 в регистр AH. В результате регистр AX будет содержать 0124.

Нет необходимости в каких-либо специальных методах программирования для получения четной или нечетной адресации, не обязательно также знать является адрес четным или нет. Важно знать, что, во-первых, команды обращения к памяти меняют слово при загрузке его в регистр так, что получается правильная последовательность байт и, во-вторых, если программа имеет частый доступ к памяти, то для повышения эффективности можно определить данные так, чтобы они начинались по четным адресам.

Например, поскольку начало сегмента должно всегда находиться по четному адресу, первое поле данных начинается также по четному адресу и пока следующие поля определены как слова, имеющие четную длину, они все будут начинаться на четных адресах. В большинстве случаев, однако, невозможно заметить ускорения работы при четной адресации из-за очень высокой скорости работы процессоров.

Ассемблер имеет директиву EVEN, которая вызывает выравнивание данных и команд на четные адреса памяти.

ПРИМЕР МАШИННЫХ КОДОВ: ОПРЕДЕЛЕНИЕ РАЗМЕРА ПАМЯТИ

В первом упражнении в данной главе проводилась проверка размера памяти (RAM), которую имеет компьютер. BIOS (базовая система ввода/вывода) в ROM имеет подпрограмму, которая определяет размер памяти. Можно обратиться в BIOS по команде INT, в данном случае по прерыванию 12H. В результате BIOS возвращает в регистр AX размер памяти в килобайтах. Загрузите в память DEBUG и введите для INT 12H и RET следующие машинные коды:

```
E CS:100 CD 12 CB
```

Нажмите R (и Return) для отображения содержимого регистров и первой команды. Регистр IP содержит 0100, при этом высвечивается команда INT 12H. Теперь нажмите T (и Return) несколько раз и просмотрите выполняемые команды BIOS (отладчик показывает мнемокоды, хотя в действительности выполняются машинные коды):

```
STI
PUSH  DS
MOV   AX,0040
MOV   DS,AX
MOV   AX,[0013]
POP   DS
IRET
```

В этот момент регистр AX содержит размер памяти в шестнадцатиричном формате. Теперь введите еще раз команду T для выхода из BIOS и возврата в вашу программу. На экране появится команда RET для машинного кода CB, который был введен вами.

СПЕЦИАЛЬНЫЕ СРЕДСТВА ОТЛАДЧИКА

В операционной системе DOS версии 2.0 и старше можно использовать DEBUG для ввода команд ассемблера так же, как и команд машинного языка. На практике можно пользоваться обоими методами.

Команда A

Команда отладчика A (Assemble) переводит DEBUG в режим приема команд ассемблера и перевода их в машинные коды. Установим начальный адрес следующим образом:

```
A 100 [Return]
```

Отладчик выдаст значение адреса сегмента кодов и смещения в виде xxxx:0100. Теперь можно вводить каждую команду, завершая клавишей Return. Когда вся программа будет введена, нажмите снова клавишу Return для выхода из режима ассемблера. Введите следующую программу:

```
MOV  AL,25     [Return]
MOV  BL,32     [Return]
ADD  AL,BL     [Return]
RET            [Return]
```

по завершению на экране будет следующая информация:

```
xxxx:0100     MOV  AL,25
xxxx:0102     MOV  BL,32
xxxx:0104     ADD  AL,BL
xxxx:0106     RET
```

В этот момент отладчик готов к приему следующей команды. При нажатии Return операция будет прекращена.

Можно видеть, что отладчик определил стартовые адреса каждой команды. Прежде чем выполнить программу, проверим сгенерированные машинные коды.

Команда U

Команда отладчика U (Unassemble) показывает машинные коды для команд ассемблера. Необходимо сообщить отладчику адреса первой и последней команды, которые необходимо просмотреть (в данном случае 100 и 106). Введите:

```
U 100,106 [и Return]
```

и на экране появится

```
xxxx:0100     B025     MOV  AL,25
xxxx:0102     B332     MOV  BL,32
xxxx:0104     00D8     ADD  AL,BL
xxxx:0106     C3       RET
```

Теперь проведем трассировку выполнения программы, начиная с команды R для вывода содержимого регистров и первой команды программы. С помощью команд T выполним последовательно все команды программы.

Теперь вы знаете, как вводить программу в машинном коде или на языке ассемблера. Обычно используется ввод на языке ассемблера, когда машинный код неизвестен, а ввод в машинном коде - для изменения программы во время выполнения. Однако в действительности программа DEBUG предназначена для отладки программ и в следующих главах основное внимание будет уделено использованию языка ассемблера.

Сохранение программы из отладчика

Можно использовать DEBUG для сохранения программ на диске в следующих случаях:

1. После загрузки программы в память машины и ее модификации необходимо сохранить измененный вариант. Для этого следует:

   - загрузить программу по ее имени: DEBUG n:имя файла [Return]
   - просмотреть программу с помощью команды D и ввести изменения по команде E,
   - записать измененную программу: W [Return]

2. Необходимо с помощью DEBUG написать небольшую по объему программу и сохранить ее на диске. Для этого следует:

   - вызвать отладчик DEBUG,
   - с помощью команд A (assemble) и E (enter) написать программу,
   - присвоить программе имя: N имя файла.COM [Return]. Тип программы должен быть COM (см. гл.6 для пояснений по COM-файлам),
   - так как только программист знает, где действительно кончается его программа, указать отладчику длину программы в байтах. В последнем примере концом программы является команда

   ```
   xxxx:0106     C3   RET
   ```

   Эта команда однобайтовая и поэтому размер программы будет равен 106 (конец) минус 100 (начало), т.е. 6.
   - запросить регистр CX командой: R CX [Return]
   - отладчик выдаст на этот запрос CX 0000 (нулевое значение)
   - указать длину программы - 6,
   - записать измененную программу: W [Return]

В обоих случаях DEBUG выдает сообщение "Writing nnnn bytes." (Запись nnnn байтов). Если nnnn равно 0, то произошла ошибка при вводе длины программы, и необходимо повторить запись снова.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

Отладчик DOS DEBUG это достаточное мощное средство, полезное для отладки ассемблерных программ. Однако следует быть осторожным с ее использованием, особенно для команды E (ввод). Ввод данных в неправильные адреса памяти или ввод некорректных данных могут привести к непредсказуемым результатам. На экране в этом случае могут появиться "странные" символы, клавиатура заблокирована или даже DOS прервет DEBUG и перезагрузит себя с диска. Какие-либо серьезные повреждения вряд ли произойдут, но возможны некоторые неожиданности, а также потеря данных, которые вводились при работе с отладчиком.

Если данные, введенные в сегмент данных или сегмент кодов, оказались некорректными, следует, вновь используя команду E, исправить их. Однако, можно не заметить ошибки и начать трассировку программы. Но и здесь возможно еще использовать команду E для изменений. Если необходимо начать выполнение с первой команды, то следует установить в регистре командного указателя (IP) значение 0100. Введите команду R (register) и требуемый регистр в следующем виде:

R IP [Return]

Отладчик выдаст на экран содержимое регистра IP и перейдет в ожидание ввода. Здесь следует ввести значение 0100 и нажать для проверки результата команду R (без IP). 0тладчик выдаст содержимое регистров, флагов и первую выполняемую команду. Теперь можно, используя команду T, вновь выполнить трассировку программы.

Если ваша программа выполняет какие-либо подсчеты, то возможно потребуется очистка некоторых областей памяти и регистров. Но убедитесь в сохранении содержимого регистров CS, DS, SP и SS, которые имеют специфическое назначение.

Прочитайте в руководстве по DOS главу о программе DEBUG. В настоящий момент рекомендуется: вводный материал и следующие команды отладчика: дамп (D), ввод (E), шестнадцатиричный (H), имя (N), выход (Q), регистры (R), трассировка (T) и запись (W). Можно ознакомиться также и с другими командами и проверить как они работают.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

2.1. Напишите машинные команды для
   а) пересылки шест. значения 4629 в регистр AX;
   б) сложения шест. 036A с содержимым регистра AX.

2.2. Предположим, что была введена следующая е команда:

E CS:100 B8 45 01 05 25 00

Вместо шест. значения 45 предполагалось 54. Напишите команду E для корректировки только одного неправильно введенного байта, т.е. непосредственно замените 45 на 54.

2.3. Предположим, что введена следующая е команда:

E CS:100 B8 04 30 05 00 30 CB

а) Что представляют собой эти команды? (Сравните с первой программой в этой главе).

б) После выполнения этой программы в регистре AX должно быть значение 0460, но в действительности оказалось 6004. В чем ошибка и

как ее исправить?

в) После исправления команд необходимо снова выполнить программу с первой команды. Какие две команды отладчика потребуются?

2.4. Имеется следующая программа в машинных кодах:

B0 25 D0 E0 B3 15 F6 E3 CB

Программа выполняет следующее:

- пересылает шест. значение 25 в регистр AL;
- сдвигает содержимое регистра AL на один бит влево (в результате в AL будет 4A);
- пересылает шест. значение 15 в регистр BL;
- умножает содержимое регистра AL на содержимое регистра BL.

Используйте отладчик для ввода (E) этой программы по адресу CS:100. Не забывайте, что все значения представлены в шестнадцатиричном виде. После ввода программы наберите D CS:100 для просмотра сегмента кода. Затем введите команду R и необходимое число команд T для пошагового выполнения программы до команды RET. Какое значение будет в регистре AX в результате выполнения программы?

2.5. Используйте отладчик для ввода (E) следующей программы в машинных кодах:

Данные: 25 15 00 00
Машинный код: A0 00 00 D0 E0 F6 26 01 00 A3 02 00 CB

Программа выполняет следующее:

- пересылает содержимое одного байта по адресу DS:00 (25) в регистр AL;
- сдвигает содержимое регистра AL влево на один бит (получая в результате 4A);
- умножает AL на содержимое одного байта по адресу DS:01 (15);
- пересылает результат из AX в слово, начинающееся по адресу DS:02.

После ввода программы используйте команды D для просмотра сегмента данных и сегмента кода. Затем введите команду R и необходимое число команд T для достижения конца программы (RET). В этот момент регистр AX должен содержать результат 0612. Еще раз используйте команду D DS:00 и заметьте, что по адресу DS:02 значение записано как 1206.

2.6. Для предыдущего задания (2.5) постройте команды для записи программы на диск под именем TRIAL.COM.

2.7. Используя команду A отладчика, введите следующую про грамму:

```
MOV  BX,25
ADD  BX,30
SHL  BX,01
SUB  BX,22
NOP
RET
```

сделайте ассемблирование и трассировку выполнения этой программы до команды NOP.

Ц е л ь: показать основные требования к программам на языке ассемблера и этапы ассемблирования, компановки и выполнения программы.

ВВЕДЕНИЕ

В гл.2 было показано как ввести и выполнить программу на машинном языке. Несомненно при этом ощутима трудность расшифровки машинного кода даже для очень небольшой программы. Сомнительно, чтобы кто-либо серьезно кодировал программы на машинном языке, за исключением разных "заплат" (корректировок) в программе на языках высокого уровня и прикладные программы. Более высоким уровнем кодирования является уровень ассемблера, на котором программист пользуется символическими мнемокодами вместо машинных команд и описательными именами для полей данных и адресов памяти.

Программа написанная символическими мнемокодами, которые используются в языке ассемблера, представляет собой исходный модуль. Для формирования исходного модуля применяют программу DOS EDLIN или любой другой подходящий экранный редактор. Затем с помощью программы ассемблерного транслятора исходный текст транслируется в машинный код, известный как объектная программа. И наконец, программа DOS LINK определяет все адресные ссылки для объектной программы, генерируя загрузочный модуль.

В данной главе объясняются требования для простой программы на ассемблере и показаны этапы ассемблирования, компановки и выполнения.

КОММЕНТАРИИ В ПРОГРАММАХ НА АССЕМБЛЕРЕ

Использование комментариев в программе улучшает ее ясность, особенно там, где назначение набора команд непонятно. Комментарий всегда начинаются на любой строке исходного модуля с символа точка с запятой (;) и ассемблер полагает в этом случае, что все символы, находящиеся справа от ; являются комментарием. Комментарий может содержать любые печатные символы, включая пробел.

Комментарий может занимать всю строку или следовать за командой на той же строке, как это показано в двух следующих примерах:

```
1.      ;Эта строка полностью является комментарием
2.      ADD AX,BX  ;Комментарий на одной строке с командой
```

Комментарии появляются только в листингах ассемблирования исходного модуля и не приводят к генерации машинных кодов, поэтому можно включать любое количество комментариев, не оказывая влияния на эффективность выполнения программы. В данной книге команды ассемблера представлены заглавными буквами, а комментарии - строчными (только для удобочитаемости).

ФОРМАТ КОДИРОВАНИЯ

Основной формат кодирования команд ассемблера имеет следующий вид:

```
[метка]   команда   [операнд(ы)]
```

Метка (если имеется), команда и операнд (если имеется) разделяются по

крайней мере одним пробелом или символом табуляции. Максимальная длина строки - 132 символа, однако, большинство предпочитают работать со строками в 80 символов (соответственно ширине экрана). Примеры кодирования:

```
            Метка     Команда   Операнд
            COUNT     DB        1    ;Имя, команда, один операнд
                    MOV       AX,0 ;Команда, два операнда
```

Метки

Метка в языке ассемблера может содержать следующие символы:

```
            Буквы:         от A до Z и от a до z
            Цифры:          от 0 до 9
            Спецсимволы:   знак вопроса (?)
                        точка (.) (только первый символ)
                        знак "коммерческое эт" (@)
                        подчеркивание (-)
                        доллар ($)
```

Первым символом в метке должна быть буква или спецсимвол. Ассемблер не делает различия между заглавными и строчными буквами. Максимальная длина метки - 31 символ. Примеры меток: COUNT, PAGE25, $E10. Рекомендуется использовать описательные и смысловые метки. Имена регистров, например, AX, DI или AL являются зарезервированными и используются только для указания соответствующих регистров. Например, в команде

```
            ADD  AX,BX
```

ассемблер "знает", что AX и BX относится к регистрам. Однако, в команде

```
            MOV  REGSAVE,AX
```

ассемблер воспримет имя REGSAVE только в том случае, если оно будет определено в сегменте данных. В прил.3 приведен список всех зарезервированных слов ассемблера.

Команда

Мнемоническая команда указывает ассемблеру какое действие должен выполнить данный оператор. В сегменте данных команда (или директива) определяет поле, рабочую область или константу. В сегменте кода команда определяет действие, например, пересылка (MOV) или сложение (ADD).

Операнд

Если команда специфицирует выполняемое действие, то операнд определяет а) начальное значение данных или б) элементы, над которыми выполняется действие по команде. В следующем примере байт COUNTER определен в сегменте данных и имеет нулевое значение:

```
        Метка     Команда   Операнд

        COUNTER   DB   0    ;Определить байт (DB) с нулевым значением
```

Команда может иметь один или два операнда, или вообще быть без операндов. Рассмотрим следующие три примера:

```
Команда Операнд     Комментарий

Нет операндов    RET              ;Вернуться
Один операнд     INC     CX       ;Увеличить CX
Два операнда     ADD     AX,12    ;Прибавить 12 к AX
```

Метка, команда и операнд не обязательно должны начинаться с какой-либо определенной позиции в строке. Однако, рекомендуется записывать их в колонку для большей удобочитаемости программы. Для этого, например, редактор DOS EDLIN обеспечивает табуляцию через каждые восемь позиций.

ДИРЕКТИВЫ

Ассемблер имеет ряд операторов, которые позволяют управлять процессом ассемблирования и формирования листинга. Эти операторы называются псевдокомандами или директивами. Они действуют только в процессе ассемблирования программы и не генерируют машинных кодов. Большинство директив показаны в следующих разделах. В гл.24 подробно описаны все директивы ассемблера и приведено более чем достаточно соответствующей информации. Гл.24 можно использовать в качестве справочника.

Директивы управления листингом: PAGE и TITLE

Ассемблер содержит ряд директив, управляющих форматом печати (или листинга). Обе директивы PAGE и TITLE можно использовать в любой программе.

Д и р е к т и в а PAGE. В начале программы можно указать количество строк, распечатываемых на одной странице, и максимальное количество символов на одной строке. Для этой цели служит директива PAGE. Следующей директивой устанавливается 60 строк на страницу и 132 символа в строке:

```
PAGE 60,132
```

Количество строк на странице может быть в пределах от 10 до 255, а символов в строке - от 60 до 132. По умолчанию в ассемблере установлено PAGE 66,80.

Предположим, что счетчик строк установлен на 60. В этом случае ассемблер, распечатав 60 строк, выполняет прогон листа на начало следующей страницы и увеличивает номер страницы на единицу. Кроме того можно заставить ассемблер сделать прогон листа на конкретной строке, например, в конце сегмента. Для этого необходимо записать директиву PAGE без операндов. Ассемблер автоматически делает прогон листа при обработке директивы PAGE.

Д и р е к т и в а TITLE. Для того, чтобы вверху каждой страницы листинга печатался заголовок (титул) программы, используется директива TITLE в следующем формате:

```
TITLE   текст
```

Рекомендуется в качестве текста использовать имя программы, под которым она находится в каталоге на диске. Например, если программа называется ASMSORT, то можно использовать это имя и описательный комментарий общей длиной до 60 символов:

TITLE ASMSORT - Ассемблерная программа сортировки имен

В ассемблере также имеется директива подзаголовка SUBTTL, которая может оказаться полезной для очень больших программ, содержащих много подпрограмм.

Директива SEGMENT

Любые ассемблерные программы содержат по крайней мере один сегмент - сегмент кода. В некоторых программах используется сегмент для стековой памяти и сегмент данных для определения данных. Ассемблерная директива для описания сегмента SEGMENT имеет следующий формат:

```
Имя      Директива     Операнд
имя      SEGMENT       [параметры]
         .
         .
         .
имя      ENDS
```

Имя сегмента должно обязательно присутствовать, быть уникальным и соответствовать соглашениям для имен в ассемблере. Директива ENDS обозначает конец сегмента. Обе директивы SEGMENT и ENDS должны иметь одинаковые имена. Директива SEGMENT может содержать три типа параметров, определяющих выравнивание, объединение и класс.

1. В ы р а в н и в а н и е. Данный параметр определяет границу начала сегмента. Обычным значением является PARA, по которому сегмент устанавливается на границу параграфа. В этом случае начальный адрес делится на 16 без остатка, т.е. имеет шест. адрес nnn0. В случае отсутствия этого операнда ассемблер принимает по умолчанию PARA.
2. О б ъ е д и н е н и е. Этот элемент определяет объединяется ли данный сегмент с другими сегментами в процессе компановки после ассемблирования (пояснения см. в следующем разделе "Компановка программы"). Возможны следующие типы объединений: STACK, COMMON, PUBLIC, AT выражение и MEMORY. Сегмент стека определяется следующим образом:

```
имя  SEGMENT   PARA   STACK
```

Когда отдельно ассемблированные программы должны объединяться компановщиком, то можно использовать типы: PUBLIC, COMMON и MEMORY. В случае, если программа не должна объединяться с другими программами, то данная опция может быть опущена.

3. К л а с с. Данный элемент, заключенный в апострофы, используется для группирования относительных сегментов при компановке:

```
имя  SEGMENT   PARA   STACK   'Stack'
```

Фрагмент программы на рис.3.1. в следующем разделе иллюстрирует директиву SEGMENT и ее различные опции.

Директива PROC

Сегмент кода содержит выполняемые команды программы. Кроме того этот сегмент также включает в себя одну или несколько процедур, определенных директивой PROC. Сегмент, содержащий только одну процедуру имеет следующий вид:

```
имя-сегмента   SEGMENT   PARA
имя-процедуры  PROC      FAR       Сегмент
         .               кода
         .               с
         .               одной
         RET               процедурой
имя-процедуры  ENDP
имя-сегмента   ENDS
```

Имя процедуры должно обязательно присутствовать, быть уникальным и удовлетворять соглашениям по именам в ассемблере. Операнд FAR указывает загрузчику DOS, что начало данной процедуры является точкой входа для выполнения программы.

Директива ENDP определяет конец процедуры и имеет имя, аналогичное имени в директиве PROC. Команда RET завершает выполнение программы и в данном случае возвращает управление в DOS.

Сегмент может содержать несколько процедур (см. гл.7).

Директива ASSUME

Процессор использует регистр SS для адресации стека, регистр DS для адресации сегмента данных и регистр CS для адресации сегмента кода. Ассемблеру необходимо сообщить назначение каждого сегмента. Для этой цели служит директива ASSUME, кодируемая в сегменте кода следующим образом:

```
Директива Операнд
ASSUME     SS:имя_стека,DS:имя_с_данных,CS:имя_с_кода
```

Например, SS:имя_стека указывает, что ассемблер должен ассоциировать имя сегмента стека с регистром SS. Операнды могут записываться в любой последовательности. Регистр ES также может присутствовать в числе операндов. Если программа не использует регистр ES, то его можно опустить или указать ES:NOTHING.

Директива END

Как уже показано, директива ENDS завершает сегмент, а директива ENDP завершает процедуру. Директива END в свою очередь полностью завершает всю программу:

```
Директива      Операнд
END           [имя_процедуры]
```

Операнд может быть опущен, если программа не предназначена для выполнения, например, если ассемблируются только определения данных, или эта программа должна быть скомпанована с другим (главным) модулем. Для обычной программы с одним модулем операнд содержит имя, указанное в директиве PROC, которое было обозначено как FAR.

ПАМЯТЬ И РЕГИСТРЫ

Рассмотрим особенности использования в командах имен, имен в квадратных скобках и чисел. В следующих примерах положим, что WORDA определяет слово в памяти:

```
MOV  AX,BX     ;Переслать содержимое BX в регистр AX
MOV  AX,WORDA  ;Переслать содержимое WORDA в регистр AX
```

```
MOV  AX,[BX]   ;Переслать содержимое памяти по адресу
               ; в регистре BX в регистр AX
MOV  AX,25     ;Переслать значение 25 в регистр AX
MOV  AX,[25]   ;Переслать содержимое по смещению 25
```

Новым здесь является использование квадратных скобок, что потребуется в следующих главах.

ИНИЦИАЛИЗАЦИЯ ПРОГРАММЫ

Существует два основных типа загрузочных программ: EXE и COM. Рассмотрим требования к EXE-программам, а COM-программы будут представлены в гл.6. DOS имеет четыре требования для инициализации ассемблерной EXE-программы: 1) указать ассемблеру, какие сегментные регистры должны соответствовать сегментам, 2) сохранить в стеке адрес, находящийся в регистре DS, когда программа начнет выполнение, 3) записать в стек нулевой адрес и 4) загрузить в регистр DS адрес сегмента данных.

Выход из программы и возврат в DOS сводится к использованию команды RET. Рис.3.1 иллюстрирует требования к инициализации и выходу из программы:

1. ASSUME - это ассемблерная директива, которая устанавливает для ассемблера соответствие между конкретными сегментами и сегментными регистрами; в данном случае, CODESG - CS, DATASG - DS и STACKSG - SS. DATASG и STACKSG не определены в этом примере, но они будут представлены следующим образом:

```
STACKSG  SEGMENT  PARA  STACK  Stack  'Stack'
DATASG   SEGMENT  PARA  'Data'
```

Ассоциируя сегменты с сегментными регистрами, ассемблер сможет определить смещения к отдельным областям в каждом сегменте. Например, каждая команда в сегменте кодов имеет определенную длину: первая команда имеет смещение 0, и если это двухбайтовая команда, то вторая команда будет иметь смещение 2 и т.д.

2. Загрузочному модулю в памяти непосредственно предшествует 256-байтовая (шест.100) область, называемая префиксом программного сегмента PSP. Программа загрузчика использует регистр DS для установки адреса начальной точки PSP. Пользовательская программа должна сохранить этот адрес, поместив его в стек. Позже, команда RET использует этот адрес для возврата в DOS.

3. В системе требуется, чтобы следующее значение в стеке являлось нулевым адресом (точнее, смещением). Для этого команда SUB очищает регистр AX, вычитая его из этого же регистра AX, а команда PUSH заносит это значение в стек.

4. Загрузчик DOS устанавливает правильные адреса стека в регистре SS и сегмента кодов в регистре CS. Поскольку программа загрузчика использует регистр DS для других целей, необходимо инициализировать регистр DS двумя командами MOV, как показано на рис.3.1. В следующем разделе этой главы "Исходная программа. Пример II" детально поясняется инициализация регистра DS.

5. Команда RET обеспечивает выход из пользовательской программы и возврат в DOS, используя для этого адрес, записанный в стек в начале программы командой PUSH DS. Другим обычно используемым выходом является команда INT 20H.

```
CODESG    SEGMENT PARA 'CODE'
```

```
    BEGIN     PROC FAR
1.          ASSUME    CS:CODESG,DS:DATASG,SS:STACKG
2.          PUSH      DS            ;Записать DS в стек
3.          SUB       AX,AX         ;Установить ноль в AX
            PUSH      AX            ;Записать ноль в стек
4.          MOV       AX,DATASG     ;Занести адрес
            MOV       DS,AX         ; DATASG в DS
            .
            .
            .
5.          RET                     ;Возврат в DOS
    BEGIN     ENDP
    CODESG    ENDS
            END       BEGIN
```

Рис.3.1. Инициализация EXE-программы.

Теперь, даже если приведенная инициализация программы до конца не понятна - не отчаивайтесь. Каждая программа фактически имеет аналогичные шаги инициализации, так что их можно дублировать всякий раз при кодировании программ.

ПРИМЕР ИСХОДНОЙ ПРОГРАММЫ

Рис.3.2 обобщает предыдущие сведения в простой исходной программе на ассемблере. Программа содержит сегмент стека - STACKSG и сегмент кода - CODESG.

```
      page    60,132
TITLE   EXASM1  (EXE) Пример регистровых операций
;-----------------------------------------------
STACKSG SEGMENT PARA SACK 'Stack'
      DB      12 DUP('STACKSEG')
STACKSG ENDS
;-----------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
      ASSUME  SS:STACKSG,CS:CODESG,DS:NOTHING
      PUSH    DS             ;Записать DS в стек
      SUB     AX,AX          ;Записать ноль
      PUSH    AX             ; в стек

      MOV     AX,0123H       ;Записать шест.0123 в AX
      ADD     AX,0025H       ;Прибавить шест.25 к AX
      MOV     BX,AX          ;Переслать AX в BX
      ADD     BX,AX          ;Прибавить BX к AX
      MOV     CX,BX          ;Переслать BX в CX
      SUB     CX,AX          ;Вычесть AX из CX
      SUB     AX,AX          ;Очистить AX
      NOP
      RET                    ;Возврат в DOS
BEGIN   ENDP                 ;Конец процедуры

CODESG  ENDS                 ;Конец сегмента
      END     BEGIN          ;Конец программы
```

Рис.3.2. Пример исходной программы на ассемблере.

STACKSG содержит один элемент DB (определить байт), который определяет 12 копий слова 'STACKSEG'. В последующих программах стек не определяется таким способом, но при использовании отладчика для просмотра ассемблированной программы на экране, данное определение помогает локализовать стек.

CODESG содержит выполняемые команды программы, хотя первая директива ASSUME не генерирует кода. Директива ASSUME назначает регистр SS для STACKSG и регистр CS для CODESG. В действительности, эта директива сообщает ассемблеру, что для адресации в STACKSG необходимо использовать адрес в регистре SS и для адресации в CODESG - адрес в регистре CS. Системный загрузчик при загрузке программы с диска в память для выполнения устанавливает действительные адреса в регистрах SS и CS. Программа не имеет сегмента данных, так как в ней нет определения данных и, соответственно, в ASSUME нет необходимости ассигновать регистр DS.

Команды, следующие за ASSUME - PUSH, SUB и PUSH выполняют стандартные действия для инициализации стека текущим адресом в регистре DS и нулевым адресом. Поскольку, обычно, программа выполняется из DOS, то эти команды обеспечивают возврат в DOS после завершения программы. (Можно также выполнить программу из отладчика, хотя это особый случай).

Последующие команды выполняют те же действия, что показаны на рис.2.1 в предыдущей главе, когда рассматривался отладчик.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Не забывайте ставить символ "точка с запятой" перед комментариями.

- Завершайте каждый сегмент директивой ENDS, каждую процедуру - директивой ENDP, а программу - директивой END.

- В директиве ASSUME устанавливайте соответствия между сегментными регистрами и именами сегментов.

- Для EXE-программ (но не для COM-программ, см. гл.6) обеспечивайте не менее 32 слов для стека, соблюдайте соглашения по инициализации стека командами PUSH, SUB и PUSH и заносите в регистр DS адрес сегмента данных.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

3.1. Какие команды заставляют ассемблер печатать заголовок в начале каждой страницы листинга и делать прогон листа?

3.2. Какие из следующих имен неправильны: а) PC_AT, б) $50, в) @$_Z, г) 34B7, д) AX?

3.3. Какое назначение каждого из трех сегментов, описанных в этой главе?

3.4. Что конкретно подразумевает директива END, если она завершает а) программу, б) процедуру, в) сегмент?

3.5. Укажите различия между директивой и командой.

3.6. Укажите различия в назначении RET и END.

3.7. Для сегментов кода, данных и стека даны имена CDSEG, DATSEG и STKSEG соответственно. Сформируйте директиву ASSUME.

3.8. Напишите три команды для инициализации стека адресом в DS и нулевым адресом.

ГЛАВА 4 Ассемблирование и выполнение программ

Ц е л ь: показать процессы ассемблирования, компановки и выполнения программ.

ВВЕДЕНИЕ

В данной главе объясняется, как ввести в компьютер исходный ассемблерный текст программы, как осуществить ассемблирование, компоновку и выполнение программы. Кроме того, показана генерация таблицы перекрестных ссылок для целей отладки.

ВВОД ПРОГРАММЫ

На рис.3.2 был показан только исходный текст программы, предназначенный для ввода с помощью текстового редактора. Теперь можно использовать DOS EDLIN или другой текстовый редактор для ввода этой программы. Если вы никогда не пользовались программой EDLIN, то именно сейчас необходимо выполнить ряд упражнений из руководства по DOS. Для запуска программы EDLIN вставьте дискету DOS в дисковод A и форматизованную дискету в дисковод B. Чтобы убедиться в наличии на дискете свободного места для исходного текста, введите CHKDSK B:. Для винчестера во всех следующих примерах следует использовать C: вместо B:. Для ввода исходной программы EXASM1, наберите команду

```
        EDLIN B:EXASM1.ASM [Return]
```

В результате DOS загрузит EDLIN в памяти и появится сообщение "New file" и приглашение "*-". Введите команду I для ввода строк, и затем наберите каждую ассемблерную команду так, как они изображены на рис.3.2. Хотя число пробелов в тексте для ассемблера не существенно, старайтесь записывать метки, команды, операнды и комментарии, выровненными в колонки, программа будет более удобочитаемая. Для этого в EDLIN используется табуляция через каждые восемь позиций.

После ввода программы убедитесь в ее правильности. Затем наберите E (и Return) для завершения EDLIN. Можно проверить наличие программы в каталоге на диске, введите

```
        DIR B:              (для всех файлов)
    или DIR B:EXASM1.ASM    (для одного файла)
```

Если предполагается ввод исходного текста большего объема, то лучшим применением будет полноэкранный редактор. Для получения распечатки программы включите принтер и установите в него бумагу. Вызовите программу PRINT (для DOS 2.0 и старше). DOS загрузит программу в память и

распечатает текст на принтере:

```
PRINT B:EXASM1.ASM [Return]
```

Программа EXASM.ASM еще не может быть выполнена - прежде необходимо провести ее ассемблирование и компоновку. В следующем разделе показана эта же программа после ассемблирования и пояснены этапы ассемблирования и получения листинга.

ПОДГОТОВКА ПРОГРАММЫ ДЛЯ ВЫПОЛНЕНИЯ

После ввода на диск исходной программы под именем EXASM1.ASM необходимо проделать два основных шага, прежде чем программу можно будет выполнить. Сначала необходимо ассемблировать программу, а затем выполнить компоновку. Программисты на языке бейсик могут выполнить программу сразу после ввода исходного текста, в то время как для ассемблера и компилярных языков нужны шаги трансляции и компоновки.

Шаг ассемблирования включает в себя трансляцию исходного кода в машинный объектный код и генерацию OBJ-модуля. Вы уже встречали примеры машинного кода в гл.2 и примеры исходного текста в этой главе.

OBJ-модуль уже более приближен к исполнительной форме, но еще не готов к выполнению. Шаг компановки включает преобразование OBJ-модуля в EXE (исполнимый) модуль, содержащий машинный код. Программа LINK, находящаяся на диске DOS, выполняет следующее:

1. Завершает формирование в OBJ-модуле адресов, которые остались неопределенными после ассемблирования. Во многих следующих программах такие адреса ассемблер отмечает как ----R.
2. Компонует, если необходимо, более одного отдельно ассемблированного модуля в одну загрузочную (выполнимую) программу; возможно две или более ассемблерных программ или ассемблерную программу с программами, написанными на языках высокого уровня, таких как Паскаль или Бейсик.
3. Инициализирует EXE-модуль командами загрузки для выполнения.

После компановки OBJ-модуля (одного или более) в EXE-модуль, можно выполнить EXE-модуль любое число раз. Но, если необходимо внести некоторые изменения в EXE-модуль, следует скорректировать исходную программу, ассемблировать ее в другой OBJ-модуль и выполнить компоновку OBJ-модуля в новый EXE-модуль. Даже, если эти шаги пока остаются непонятными, вы обнаружите, что, получив немного навыка, весь процесс подготовки EXE-модуля будет доведен до автоматизма. Заметьте: определенные типы EXE-программ можно преобразовать в очень эффективные COM-программы. Предыдущие примеры, однако, не совсем подходят для этой цели. Данный вопрос рассматривается в главе 6.

АССЕМБЛИРОВАНИЕ ПРОГРАММЫ

Для того, чтобы выполнить исходную ассемблерную программу, необходимо прежде провести ее ассемблирование и затем компоновку. На дискете с ассемблерным пакетом имеются две версии ассемблера. ASM.EXE - сокращенная версия с отсутствием некоторых незначительных возможностей и MASM.EXE - полная версия. Если размеры памяти позволяют, то используйте версию MASM (подробности см. в соответствующем руководстве по ассемблеру).

Для ассемблирования, вставьте ассемблерную дискету в дисковод A, а дискету с исходной программой в дисковод B. Кто имеет винчестер могут использовать в следующих примерах C вместо A и B. Простейший вариант

вызова программы это ввод команды MASM (или ASM), что приведет к загрузке программы ассемблера с диска в память. На экране появится:

```
source filename [.ASM]:
object filename [filename.OBJ]:
source listing [NUL.LST]:
cross-reference [NUL.CRF]:
```

Курсор при этом расположится в конце первой строки, где необходимо указать имя файла. Введите номер дисковода (если он не определен умолчанием) и имя файла в следующем виде: B:EXASM1. Не следует набирать тип файла ASM, так как ассемблер подразумевает это.

Во-втором запросе предполагается аналогичное имя файла (но можно его заменить). Если необходимо, введите номер дисковода B:.

Третий запрос предполагает, что листинг ассемблирования программы не требуется. Для получения листинга на дисководе B наберите B: и нажмите Return.

Последний запрос предполагает, что листинг перекрестных ссылок не требуется. Для получения листинга на дисководе B, наберите B: и нажмите Return.

Если вы хотите оставить значения по умолчанию, то в трех последних запросах просто нажмите Return. Ниже приведен пример запросов и ответов, в результате которых ассемблер должен создать OBJ, LST и CRF-файлы. Введите ответы так, как показано, за исключением того, что номер дисковода может быть иной.

```
source filename [.ASM]:B:EXASM1 [Return]
object filename [filename.OBJ]:B: [Return]
source listing [NUL.LST]:B: [Return]
cross-reference [NUL.CRF]:B: [Return]
```

Всегда необходимо вводить имя исходного файла и, обычно, запрашивать OBJ-файл - это требуется для компановки программы в загрузочный файл. Возможно потребуется указание LST-файла, особенно, если необходимо проверить сгенерированный машинный код. CRF-файл полезен для очень больших программ, где необходимо видеть, какие команды ссылаются на какие поля данных. Кроме того, ассемблер генерирует в LST-файле номера строк, которые используются в CRF-файле.

В прил.4 "Режимы ассемблирования и редактирования" перечислены режимы (опции) для ассемблера версий 1.0 и 2.0.

Ассемблер преобразует исходные команды в машинный код и выдает на экран сообщения о возможных ошибках. Типичными ошибками являются нарушения ассемблерных соглашений по именам, неправильное написание команд (например, MOVE вместо MOV), а также наличие в операндах неопределенных имен. Программа ASM выдает только коды ошибок, которые объяснены в руководстве по ассемблеру, в то время как программа MASM выдает и коды ошибок, и пояснения к ним. Всего имеется около 100 сообщений об ошибках.

Ассемблер делает попытки скорректировать некоторые ошибки, но в любом случае следует перезагрузить текстовый редактор, исправить исходную программу (EXASM1.ASM) и повторить ассемблирование.

На рис.4.1 показан листинг, полученный в результате ассемблирования программы и записанный на диск под именем EXASM1.LST.

```
1                        page   60,132
2                  TITLE   EXASM1 (EXE) Пример регистровых операций
3                  ;-------------------------------------------
4  0000              STACKSG SEGMENT PARA SACK 'Stack'
5  0000  0C [                DB      12 DUP('STACKSEG')
6          53 54 41 43
```

```
7          4B 53 45 47
8                    ]
9
10 0060              STACKSG ENDS
11                  ;--------------------------------------------
12 0000              CODESG  SEGMENT PARA 'Code'
13 0000              BEGIN   PROC    FAR
14                       ASSUME  SS:STACKSG,CS:CODESG,DS:NOTHING
15 0000  1E                  PUSH    DS      ;Записать DS в стек
16 0001  2B C0               SUB     AX,AX    ;Записать ноль
17 0003  50                  PUSH    AX      ; в стек
18
19 0004  B8 0123              MOV     AX,0123H ;Записать шест.0123 в AX
20 0007  05 0025              ADD     AX,0025H ;Прибавить шест.25 к AX
21 000A  8B D8                MOV     BX,AX    ;Переслать AX в BX
22 000C  03 D8                ADD     BX,AX    ;Прибавить BX к AX
23 000E  8B CB                MOV     CX,BX    ;Переслать BX в CX
24 0010  2B C8                SUB     CX,AX    ;Вычесть AX из CX
25 0012  2B C0                SUB     AX,AX    ;Очистить AX
26 0014  90                  NOP
27 0015  CB                  RET            ;Возврат в DOS
28 0016              BEGIN   ENDP            ;Конец процедуры
29
30 0016              CODESG  ENDS            ;Конец сегмента
31                       END     BEGIN   ;Конец программы
-----------------------------------------------------------------------
  Segments and Groups:
          N a m e         Size   Align    Combine   Class
  CODESG . . . . . . . . . . . . 0016   PARA     NONE      'CODE'
  STACKSG. . . . . . . . . . . . 0060   PARA     STACK     'STACK'

  Symbols:
          N a m e         Type   Value    Attr
  BEGIN. . . . . . . . . . . . . F PROC 0000    CODESG   Length=0016
```

Рис.4.1. Листинг ассемблирования программы.

В начале листинга обратите внимание на реакцию ассемблера на директивы PAGE и TITLE. Никакие директивы, включая SEGMENT, PROC, ASSUME и END не генерируют машинных кодов.

Листинг содержит не только исходный текст, но также слева транслированный машинный код в шестнадцатиричном формате. В самой левой колонке находится шест.адреса команд и данных.

Сегмент стека начинается с относительного адреса 0000. В действительности он загружается в память в соответствии с адресом в регистре SS и нулевым смещением относительно этого адреса. Директива SEGMENT устанавливает 16-кратный адрес и указывает ассемблеру, что это есть начало стека. Сама директива не генерирует машинный код. Команда DB, также находится по адресу 0000, содержит 12 копий слова 'STACKSEG'; машинный код представлен шест.0C (десятичное 12) и шест. представлением ASCII символов. (В дальнейшем можно использовать отладчик для просмотра результатов в памяти). Сегмент стека заканчивается по адресу шест.0060, который эквивалентен десятичному значению 96 (12x8).

Сегмент кода также начинается с относительного адреса 0000. Он загружается в память в соответствии с адресом в регистре CS и нулевым смещением относительно этого адреса. Поскольку ASSUME является директивой ассемблеру, то первая команда, которая генерирует действительный машинный код есть PUSH DS - однобайтовая команда (1E), находящаяся на нулевом смещении. Следующая команда SUB AX,AX генерирует двухбайтовый машинный код

(2B C0), начинающийся с относительного адреса 0001. Пробел между байтами только для удобочитаемости. В данном примере встречаются одно-, двух- и трехбайтовые команды.

Последняя команда END содержит операнд BEGIN, который имеет отношение к имени команды PROC по смещению 0000. Это есть адрес сегмента кодов, с которого начинается выполнение после загрузки программы.

Листинг ассемблирования программы EXASM1.LST, имеет по директиве PAGE ширину 132 символа и может быть распечатан. Многие принтеры могут печатать текст сжатым шрифтом. Включите ваш принтер и введите команду

```
MODE LPT1:132,6
```

Таблица идентификаторов

За листингом ассемблирования программы следует таблица идентификаторов. Первая часть таблицы содержит определенные в программе сегменты и группы вместе с их размером в байтах, выравниванием и классом. Вторая часть содержит идентификаторы - имена полей данных в сегменте данных (в нашем примере их нет) и метки, назначенные командам в сегменте кодов (одна в нашем примере). Для того, чтобы ассемблер не создавал эту таблицу, следует указать параметр /N вслед за командой MASM, т.е. MASM/N.

Двухпроходный ассемблер

В процессе трансляции исходной программы ассемблер делает два просмотра исходного текста, или два прохода. Одной из основных причин этого являются ссылки вперед, что происходит в том случае, когда в некоторой команде кодируется метка, значение которой еще не определено ассемблером.

В первом проходе ассемблер просматривает всю исходную программу и строит таблицу идентификаторов, используемых в программе, т.е. имен полей данных и меток программы и их относительных адресов в программе. В первом проходе подчитывается объем объектного кода, но сам объектный код не генерируется.

Во втором проходе ассемблер использует таблицу идентификаторов, построенную в первом проходе. Так как теперь уже известны длины и относительные адреса всех полей данных и команд, то ассемблер может сгенерировать объектный код для каждой команды. Ассемблер создает, если требуется, файлы: OBJ, LST и CRF.

КОМПАНОВКА ПРОГРАММЫ

Если в результате ассемблирования не обнаружено ошибок, то следующий шаг - компановка объектного модуля. Файл EXASM1.OBJ содержит только машинный код в шестнадцатеричной форме. Так как программа может загружаться почти в любое место памяти для выполнения, то ассемблер может не определить все машинные адреса. Кроме того, могут использоваться другие (под) программы для объединения с основной. Назначением программы LINK является завершение определения адресных ссылок и объединение (если требуется) нескольких программ.

Для компановки ассемблированной программы с дискеты, вставьте дискету DOS в дисковод A, а дискету с программой в дисковод B. Пользователи винчестерского диска могут загрузить компоновщик LINK прямо с дисковода C. Введите команду LINK и нажмите клавишу Return. После загрузки в память, компоновщик выдает несколько запросов (аналогично MASM), на которые необходимо ответить:

```
Запрос компоновщика        Ответ          Действие

Object Modules [.OBJ]:     B:EXASM1  Компонует EXASM1.OBJ
Run file [EXASM1.EXE]:      B:        Создает EXASM1.EXE
List file [NUL.MAP]:       CON       Создает EXASM1.MAP
Libraries [.LIB]:          [Return]  По умолчанию
```

Первый запрос - запрос имен объектных модулей для компановки, тип OBJ можно опустить.

Второй запрос - запрос имени исполнимого модуля (файла), (по умолчанию A:EXASM1.EXE). Ответ B: требует, чтобы компоновщик создал файл на дисководе B. Практика сохранения одного имени (при разных типах) файла упрощает работу с программами.

Третий запрос предполагает, что LINK выбирает значение по умолчанию - NUL.MAP (т.е. MAP отсутствует). MAP-файл содержит таблицу имен и размеров сегментов и ошибки, которые обнаружит LINK. Типичной ошибкой является неправильное определение сегмента стека. Ответ CON предполагает, что таблица будет выведена на экран, вместо записи ее на диск. Это позволяет сэкономить место в дисковой памяти и сразу просмотреть таблицу непосредственно на экране. В нашем примере MAP-файл содержит следующую информацию:

```
Start    Stop     Length   Name

00000H   00015H   0016H    CODESG
00020H   0007FH   0060H    STACKSG
```

Для ответа на четвертый запрос - нажмите Return, что укажет компоновщику LINK принять остальные параметры по умолчанию. Описание библиотечных средств можно найти в руководстве по DOS.

На данном этапе единственной возможной ошибкой может быть указание неправильных имен файлов. Исправить это можно только перезапуском программы LINK. В прил.4 перечислен ряд режимов компоновщика LINK.

ВЫПОЛНЕНИЕ ПРОГРАММЫ

После ассемблирования и компановки программы можно (наконец-то!) выполнить ее. На рис.4.2 приведена схема команд и шагов для ассемблирования, компановки и выполнения программы EXASM1. Если EXE-файл находится на дисководе B, то выполнить ее можно командой:

B:EXASM1.EXE   или   B:EXASM1

DOS предполагает, что файл имеет тип EXE (или COM), и загружает файл для выполнения. Но так как наша программа не вырабатывает видимых результатов, выполним ее трассировкой под отладчиком DEBUG. Введите

DEBUG B:EXASM1.EXE

В результате DOS загрузит программу DEBUG, который, в свою очередь, загрузит требуемый EXE-модуль. После этого отладчик выдаст дефис (-) в качестве приглашения. Для просмотра сегмента стека введите

D SS:0

Эту область легко узнать по 12-кратному дублированию константы STACKSEG. Для просмотра сегмента кода введите

D CS:0

Сравните машинный код с листингом ассемблера:

```
1E2BC050B823010525008BD803 ...
```

Непосредственные операнды, приведенные в листинге ассемблирования как 0123 и 0025 в памяти представлены в виде 2301 и 2500 соответственно. В данном случае листинг ассемблирования не вполне соответствует машинному коду. Все двухбайтовые адреса (слова) и непосредственные операнды в машинном коде хранятся в обратном порядке.

Рис.4.2. Схема ассемблирования, компановки и выполнения программы.

Введите R для просмотра содержимого регистров и выполните программу с помощью команды T (трассировка). Обратите внимание на воздействие двух команд PUSH на стек - в вершине стека теперь находится содержимое регистра DS и нулевой адрес.

В процессе пошагового выполнения программы обратите внимание на содержимое регистров. Когда вы дойдете до команды RET, можно ввести Q (Quit - выход) для завершения работы отладчика.

Используя команду dir, можно проверить наличие ваших файлов на диске:

```
DIR B:EXASM1.*
```

В результате на экране появится следующие имена файлов: EXASM1.BAK (если для корректировки EXASM1.ASM использовался редактор EDLIN), EXASM1.ASM, EXASM1.OBJ, EXASM1.LST, EXASM1.EXE и EXASM1.CRF. Последовательность этих файлов может быть иной в зависимости от того, что уже находится на диске.

Очевидно, что разработка ряда программ приведет к занятию дискового пространства. Для проверки оставшегося свободного места на диске полезно использовать команду DOS CHKDSK. Для удаления OBJ-, CRF-, BAK- и LST-файлов с диска следует использовать команду ERASE (или DEL):

```
ERASE B:EXASM1.OBJ, ...
```

Следует оставить (сохранить) ASM-файл для последующих изменений и EXE-файл для выполнения.

В следующем разделе представлено определение данных в сегменте данных. Позже будет описана таблица перекрестных ссылок.

ПРИМЕР ИСХОДНОЙ ПРОГРАММЫ

Особенность программы, приведенной на рис.4.1, состоит в том, что она не содержит определения данных. Обычно все программы имеют определенные константы, рабочие поля для арифметических вычислений и области для операций ввода-вывода.

В главе 2 (рис.2.3) была рассмотрена программа в машинных кодах, в которой были определены два поля данных. В этой главе на рис.4.3 приводится аналогичная программа, но на этот раз написанная на языке ассемблера и для краткости уже ассемблированная. Эта программа знакомит с несколькими новыми особенностями.

```
1                           page   60,132
2                     TITLE   EXASM2 (EXE) Операции пересылки и сложения
3                     ;-------------------------------------------
4  0000               STACKSG SEGMENT PARA SACK 'Stack'
5  0000  20 [                  DB      32 DUP(?)
6        ????
7            ]
8
9  0040               STACKSG ENDS
10                    ;-------------------------------------------
11 0000               DATASG  SEGMENT PARA 'Data'
12 0000  00FA            FLDA    DW      250
13 0002  007D            FLDB    DW      125
14 0004  ????          FLDC    DW      ?
15 0006               DATASG  ENDS
16                    ;-------------------------------------------
17 0000               CODESG  SEGMENT PARA 'Code'
18 0000               BEGIN   PROC    FAR
19                         ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG,ES:NO
THING
20 0000  1E                  PUSH    DS          ;Записать DS в стек
21 0001  2B C0                 SUB     AX,AX       ;Записать в стек
22 0003  50                  PUSH    AX          ; нулевой адрес
23 0004  B8 ---- R             MOV     AX,DATASG   ;Поместить адрес DATASG
24 0007  8E D8                 MOV     DS,AX       ; в регистр DS
25
26 0009  A1 0000 R             MOV     AX,FLDA     ;Переслать 0250 в AX
27 000C  03 06 0002 R          ADD     AX,FLDB     ;Прибавить 0125 к AX
28 0010  A3 0004 R             MOV     FLDC,AX     ;Записать сумму в FLDC
29 0013  CB                  RET                 ;Вернуться в DOS
30 0014               BEGIN   ENDP
31 0014               CODESG  ENDS
32                         END     BEGIN
--------------------------------------------------------------------
   Segments and Groups:
            N a m e          Size   Align    Combine   Class
   CODESG . . . . . . . . . . . . 0014   PARA     NONE      'CODE'
   DATASG . . . . . . . . . . . . 0006   PARA     NONE      'DATA'
   STACKSG. . . . . . . . . . . . 0040   PARA     STACK     'STACK'

   Symbols:
            N a m e          Type   Value    Attr
   BEGIN. . . . . . . . . . . . . F PROC 0000     CODESG   Length=0014
   FLDA . . . . . . . . . . . . . L WORD         0000     DATASG
   FLDB . . . . . . . . . . . . . L WORD         0002     DATASG
   FLDC . . . . . . . . . . . . . L WORD         0004     DATASG
```

Рис.4.3. Листинг ассемблирования программы с сегментом данных.

Сегмент стека содержит директиву DW (Define Word - определить слово), описывающая 32 слова, в которых генерируется неопределенное значение обозначенное знаком вопроса (?). Определение размера стека в 32 слова является наиболее реальным, так как в больших программах может потребоваться много "прерываний" для ввода-вывода и вызовов подпрограмм - все они используют стек. Определение стека дублированием константы 'STACKSEG' в примере на рис.3.2 необходимо лишь для удобства при работе с отладчиком DEBUG.

З а м е ч а н и е: Определяйте размер стека не менее 32 слов. При

малых размерах стека ни ассемблер, ни компоновщик не смогут определить этого и выполнение программы может разрушиться самым непредсказуемым образом.

В примере на рис.4.3 определен сегмент данных DATASG, начинающийся по относительному адресу 0000. Этот сегмент содержит три значения в формате DW. Поле FLDA определяет слово (два байта), содержащее десятичное значение 250, которое ассемблер транслирует в шест.00FA (см. на рисунке слева).

Поле FLDB определяет слово с десятичным значением 125, которое транслируется в шест.007D. Действительные значения этих двух констант в памяти - FA00 и 7D00 соответственно, что можно проверить с помощью отладчика DEBUG.

Поле FLDC определяет слово с неизвестным значением, обозначенным знаком вопроса (?).

Сегмент кода в данном примере имеет имя CODESG и отличается новыми особенностями, связанными с сегментом данных. Во-первых, директива ASSUME указывает на определение DATASG через регистр DS. Данной программе не требуется регистр ES, но некоторые программисты описывают его для стандартизации. Во-вторых, после команд PUSH, SUB и PUSH, которые инициализируют стек, следуют две команды, обеспечивающие адресацию сегмента данных:

```
0004  B8 ---- R      MOV  AX,DATASG
0007  8E D8          MOV  DS,AX
```

Первая команда MOV загружает DATASG в регистр AX. Конечно, на самом деле команда не может загрузить сегмент в регистр - она загружает лишь адрес сегмента DATASG. Обратите внимание на машинный код слева:

```
B8 ---- R
```

Четыре дефиса говорят о том, что ассемблер не может определить адрес DATASG; он определяется лишь когда объектная программа будет скомпонована и загружена для выполнения. Поскольку загрузчик может расположить программу в любом месте памяти, ассемблер оставляет данный адрес открытым и показывает это символом R; компоновщик должен будет подставить в это место действительный адрес.

Вторая команда MOV пересылает содержимое регистра AX в регистр DS. Таким образом, данная программа имеет директиву ASSUME, которая соотносит регистр DS с сегментом данных, и команды, инициализирующие регистр DS относительным адресом DATASG.

Могут возникнуть два вопроса по поводу этой программы. Во-первых, почему не использовать одну команду для инициализации регистра DS, например

```
MOV  DS,DATASG ?
```

Дело в том, что не существует команд для непосредственной пересылки данных из памяти в регистр DS. Следовательно, для инициализации DS необходимо кодировать две команды.

Во-вторых, почему программа инициализирует регистр DS, а регистры SS и CS нет? Оказывается, регистры SS и CS инициализируются автоматически при загрузке программы для выполнения, а ответственность за инициализацию регистра DS и, если требуется ES, лежит полностью на самой программе.

Пока все эти требования могут показаться весьма туманными, но сейчас нет необходимости понимать их. Все последующие программы используют аналогичную стандартную инициализацию стека и сегмента данных. Поэтому можно просто копировать данные коды для каждой новой программы. Действительно, вы можете сохранить на диске стандартную часть программы и для каждой новой программы копировать эту часть с новым именем, и, используя затем редактор, записать дополнительные команды.

В качестве упражнения, создайте с помощью вашего редактора программу,

приведенную на рис.4.3, выполните ее ассемблирование и компоновку. Затем с помощью отладчика DEBUG просмотрите сегмент кодов, сегмент данных, регистры и проделайте пошаговое выполнение программы.

ФАЙЛ ПЕРЕКРЕСТНЫХ ССЫЛОК

В процессе трансляции ассемблер создает таблицу идентификаторов (CRF), которая может быть представлена в виде листинга перекрестных ссылок на метки, идентификаторы и переменные в программе. Для получения данного фала, необходимо на четвертый запрос ассемблера, ответить B:, полагая, что файл должен быть создан на диске B:

```
cross-reference [NUL.CRF]:B: [Return]
```

Далее необходимо преобразовать полученный CRF-файл в отсортированную таблицу перекрестных ссылок. Для этого на ассемблерном диске имеется соответствующая программа. После успешного ассемблирования введите команду CREF. На экране появится два запроса:

```
Cref filename [.CRF]:
List filename [cross-ref.REF]:
```

На первый запрос введите имя CRF-файла, т.е. B:EXASM1. На второй запрос можно ввести только номер дисковода и получить имя по умолчанию. Такой выбор приведет к записи CRF в файл перекрестных ссылок по имени EXASM1.REF на дисководе B.

Для распечатки файла перекрестных ссылок используйте команду DOS PRINT. В приложении 4 приведен ряд режимов программы CREF.

```
EXASM2  (EXE) Операции пересылки и сложения

Symbol Cross Reference    (# is definition)     Cref-1

BEGIN. . . . . . . . . . . .    18#    30     32

CODE . . . . . . . . . . . .    17
CODESG . . . . . . . . . . .    17#    19     31

DATA . . . . . . . . . . . .    11
DATASG . . . . . . . . . . .    11#    15     19    23

FLDA . . . . . . . . . . . .    12#    26
FLDB . . . . . . . . . . . .    13#    27
FLDC . . . . . . . . . . . .    14#    28

STACK. . . . . . . . . . . .     4
STACKSG. . . . . . . . . . .     4#     9     19

10 Symbols
```

Рис.4.4. Таблица перекрестных ссылок.

На рис.4.4 показана таблица перекрестных ссылок для программы, приведенной на рис.4.3. Все идентификаторы в таблице представлены в алфавитном порядке и для каждого из них указаны номера строк в исходной

программе, где они определены и имеют ссылки. Имена сегментов и элементов данных представлены в алфавитном порядке. Первое число справа в формате n# указывает на номер строки в LST-файле, где определен соответствующий идентификатор. Еще правее находятся числа, указывающие на номера строк, где имеются ссылки на этот идентификатор. Например, CODESG определен в строке 17 и имеет ссылки на строках 19 и 32.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Ассемблер преобразует исходную программу в OBJ-файл, а компоновщик - OBJ-файл в загрузочный EXE-файл.

- Внимательно проверяйте запросы и ответы на них для программ (M)ASM, LINK и CREF прежде чем нажать клавишу Return. Будьте особенно внимательны при указании дисковода.

- Программа CREF создает распечатку перекрестных ссылок.

- Удаляйте ненужные файлы с вашего диска. Регулярно пользуйтесь программой CHKDSK для проверки свободного места на диске. Кроме того периодически создавайте резервные копии вашей программы, храните резервную дискету и копируйте ее заново для последующего программирования.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

4.1. Введите команду MASM и ответьте на запросы для ассемблирования программы по имени TEMPY.ASM с получением файлов LST, OBJ и CRF, полагая, что дискета с программой находится на дисководе B.

4.2. Введите команды для программы TEMPY (из вопроса 4.1) а) для выполнения через отладчик DEBUG, б) для непосредственного выполнения из DOS.

4.3. Объясните назначение каждого из следующих файлов: а) file.BAK, б) file.ASM, в) file.LST, г) file.CRF, д) file.OBJ, е) file.EXE, ж) file.MAP.

4.4. Напишите две команды для инициализации регистра DS, полагая, что имя сегмента данных - DATSEG.

4.5. Составьте ассемблерную программу для:

- пересылки шест.30 (непосредственное значение) в регистр AL;
- сдвига содержимого регистра AL на один бит влево (команда SHL)
- пересылки шест.18 (непосредственное значение) в регистр BL;
- умножения регистра AL на BL (команда MUL BL).

Не забывайте команду RET. В программе нет необходимости определять и инициализировать сегмент данных. Не забывайте также копировать стандартную часть программы (основу программы) и использовать редактор для ее развития. Выполните ассемблирование и компоновку. Используя отладчик DEBUG, проверьте сегмент кодов, регистры и проделайте пошаговое выполнение (трассировку) программы.

4.6. Модифицируйте программу из вопроса 4.5 для:

- определения однобайтовых элементов (директива DB) по имени

FLDA, содержащего шест.28, и по имени FLDB, содержащего шест.14;
- определения двухбайтового элемента (директива DW) по имени FLDC, не имеющего значения;
- пересылки содержимого поля FLDA в регистр AL и сдвига на один бит;
- умножения содержимого регистра AL на значение в поле FLDB (MUL FLDB);
- пересылки результата из регистра AX в поле FLDC.

Для данной программы необходим сегмент данных. Выполните ассемблирование, компоновку программы и тестирование с помощью отладчика DEBUG.

ГЛАВА 5 Определение данных

Ц е л ь: Показать методам определения констант и рабочих полей в ассемблерной программе.

ВВЕДЕНИЕ

Сегмент данных предназначен для определения констант, рабочих полей и областей для ввода-вывода. В соответствии с имеющимися директивами в ассемблере разрешено определение данных различной длины: например, директива DB определяет байт, а директива DW определяет слово. Элемент данных может содержать непосредственное значение или константу, определенную как символьная строка или как числовое значение.

Другим способом определения константы является непосредственное значение, т.е. указанное прямо в ассемблерной команде, например:

```
MOV  AL,20H
```

В этом случае шестнадцатеричное число 20 становится частью машинного объектного кода. Непосредственное значение ограничено одним байтом или одним словом, но там, где оно может быть применено, оно является более эффективным, чем использование константы.

ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ

Ассемблер обеспечивает два способа определения данных: во-первых, через указание длины данных и, во-вторых, по их содержимому. Рассмотрим основной формат определения данных:

```
[имя]    Dn   выражение
```

Имя элемента данных не обязательно (это указывается квадратными скобками), но если в программе имеются ссылки на некоторый элемент, то это делается посредством имени. Правила написания имен приведены в разделе "Формат кодирования" в главе 3.

Для определения элементов данных имеются следующие директивы: DB (байт), DW (слово), DD (двойное слово), DQ (учетверенное слово) и DT (десять байт).

Выражение может содержать константу, например:

```
        FLD1     DB   25
```

или знак вопроса для неопределенного значения, например

```
        FLDB     DB   ?
```

Выражение может содержать несколько констант, разделенных запятыми и ограниченными только длиной строки:

```
        FLD3     DB   11, 12, 13, 14, 15, 16, ...
```

Ассемблер определяет эти константы в виде последовательности смежных байт. Ссылка по имени FLD3 указывает на первую константу, 11, по FLD3+1 - на вторую, 12. (FLD3 можно представить как FLD3+0). Например команда

```
        MOV  AL,FLD3+3
```

загружает в регистр AL значение 14 (шест. 0E). Выражение допускает также повторение константы в следующем формате:

```
        [имя]    Dn  число-повторений DUP (выражение) ...
```

Следующие три примера иллюстрируют повторение:

```
        DW   10 DUP(?)      ;Десять неопределенных слов
        DB   5 DUP(14)      ;Пять байт, содержащих шест.14
        DB   3 DUP(4 DUP(8));Двенадцать восьмерок
```

В третьем примере сначала генерируется четыре копии десятичной 8 (8888), и затем это значение повторяется три раза, давая в результате двенадцать восьмерок.

Выражение может содержать символьную строку или числовую константу.

Символьные строки

Символьная строка используются для описания данных, таких как, например, имена людей или заголовки страниц. Содержимое строки отмечается одиночными кавычками, например, 'PC' или двойными кавычками - "PC". Ассемблер переводит символьные строки в объектный код в обычном формате ASCII.

Символьная строка определяется только директивой DB, в которой указывается более двух символов в нормальной последовательности слева направо. Следовательно, директива DB представляет единственно возможный формат для определения символьных данных. На рис.5.1 приведен ряд примеров.

```
                        page    60,132
                   TITLE   EXDEF   (EXE) Определение данных
0000                 DATASG  SEGMENT PARA 'Data'

                   ;            Определение байта - DB:
                   ;            ----------------------
0000  ??              FLD1DB  DB     ?        ;Не инициализировано
0001  50 65 72 73 6F 6E  FLD2DB  DB     'Personal Computer' ;Сим. строка
      61 6C 20 43 6F 6D
      70 75 74 65 72
0012  20              FLD3DB  DB     32       ;Десятичная константа
0013  20              FLD4DB  DB     20H      ;Шест. константа
```

```
0014  59              FLD5DB  DB     01011001B ;Двоичная константа
0015  01 4A 41 4E 02 46  FLD6DB  DB     01,'JAN',02,'FEB',03,'MAR' ;Таб-ца
     45 42 03 4D 41 52
0021  33 32 36 35 34     FLD7DB  DB     '32654'   ;Символьные числа
0026     0A [ 00 ]       FLD8DB  DB     10 DUP(0) ;Десять нулей

                   ;            Определение слова - DW:
                   ;            ----------------------
0030  FFF0             FLD1DW  DW     0FFF0H    ;Шест. константа
0032  0059             FLD2DW  DW     01011001B ;Двоичная константа
0034  0021 R           FLD3DW  DW     FLD7DB    ;Адресная константа
0036  0003 0004 0007     FLD4DW  DW     3,4,7,8,9 ;Пять констант
     0008 0009
0040     05 [ 0000 ]     FLD5DW  DW     5 DUP(0)  ;Пять нулей

                   ;            Определение двойного слова - DD:
                   ;            -------------------------------
004A  ????????         FLD1DD  DD     ?         ;Не инициализировано
004E  43 50 00 00        FLD2DD  DD     'PC'      ;Символьная строка
0052  3C 7F 00 00        FLD3DD  DD     32572     ;Десятичное значение
0056  11 00 00 00        FLD4DD  DD     FLD3DB - FLD2DB ;Разность адресов
005A  0E 00 00 00 31 00  FLD5DD  DD     14,49     ;Две константы
     00 00
                   ;            Определ. учетверенного слова - DQ:
                   ;            ---------------------------------
0062  ????????????????   FLD1DQ  DQ     ?         ;Не инициализировано
006A  47 4D 00 00 00 00  FLD2DQ  DQ     04D47H    ;Шест. константа
     00 00
0072  3C 7F 00 00 00 00  FLD3DQ  DQ     32572     ;Десятич. константа
     00 00
                   ;            Определение десяти байт - DT:
                   ;            ----------------------------
007A  ?????????????????? FLD1DT  DT     ?         ;Не инициализировано
     ??
0084  43 50 00 00 00 00  FLD2DT  DT     'PC'      ;Символьная строка
     00 00 00 00

008E                DATASG  ENDS
                            END
---------------------------------------------------------------------
Segments and Groups:
          N a m e    Size   Align   Combine   Class
DATASG . . . . . . . . . 008E   PARA     NONE     'DATA'

Symbols:
          N a m e      Type          Value    Attr
FLD1DB . . . . . . . . . L BYTE        0000     DATASG
FLD1DD . . . . . . . . . L DWORD       004A     DATASG
FLD1DQ . . . . . . . . . L QWORD       0062     DATASG
FLD1DT . . . . . . . . . L TBYTE       007A     DATASG
FLD1DW . . . . . . . . . L WORD        0030     DATASG
FLD2DB . . . . . . . . . L BYTE        0001     DATASG
FLD2DD . . . . . . . . . L DWORD       004E     DATASG
FLD2DQ . . . . . . . . . L QWORD       006A     DATASG
FLD2DT . . . . . . . . . L TBYTE       0084     DATASG
FLD2DW . . . . . . . . . L WORD        0032     DATASG
FLD3DB . . . . . . . . . L BYTE        0012     DATASG
FLD3DD . . . . . . . . . L DWORD       0052     DATASG
FLD3DQ . . . . . . . . . L QWORD       0072     DATASG
FLD3DW . . . . . . . . . L WORD        0034     DATASG
FLD4DB . . . . . . . . . L BYTE        0013     DATASG
```

```
FLD4DD . . . . . . . . . L DWORD        0056     DATASG
FLD4DW . . . . . . . . . L WORD         0036     DATASG
FLD5DB . . . . . . . . . L BYTE         0014     DATASG
FLD5DD . . . . . . . . . L DWORD        005A     DATASG
FLD5DW . . . . . . . . . L WORD         0040     DATASG  Length =0005
FLD6DB . . . . . . . . . L BYTE         0015     DATASG
FLD7DB . . . . . . . . . L BYTE         0021     DATASG
FLD8DB . . . . . . . . . L BYTE         0026     DATASG  Length =000A
```

Рис.5.1. Определение символьных строк и числовых величин.

Числовые константы

Числовые константы используются для арифметических величин и для адресов памяти. Для описания константы кавычки не ставятся. Ассемблер преобразует все числовые константы в шестнадцатеричные и записывает байты в объектном коде в обратной последовательности - справа налево. Ниже показаны различные числовые форматы.

Д е с я т и ч н ы й   ф о р м а т. Десятичный формат допускает десятичные цифры от 0 до 9 и обозначается последней буквой D, которую можно не указывать, например, 125 или 125D. Несмотря на то, что ассемблер позволяет кодирование в десятичном формате, он преобразует эти значения в шест. объектный код. Например, десятичное число 125 преобразуется в шест. 7D.

Ш е с т н а д ц а т и р и ч н ы й   ф о р м а т. Шестнадцатиричный формат допускает шест. цифры от 0 до F и обозначается последней буквой H. Так как ассемблер полагает, что с буквы начинаются идентификаторы, то первой цифрой шест. константы должна быть цифра от 0 до 9. Например, 2EH или 0FFFH, которые ассемблер преобразует соответственно в 2E и FF0F (байты во втором примере записываются в объектный код в обратной последовательности).

Д в о и ч н ы й   ф о р м а т. Двоичный формат допускает двоичные цифры 0 и 1 и обозначается последней буквой B. Двоичный формат обычно используется для более четкого представления битовых значений в логических командах AND, OR, XOR и TEST. Десятичное 12, шест. C и двоичное 1100B все генерируют один и тот же код: шест. 0C или двоичное 0000 1100 в зависимости от того, как вы рассматриваете содержимое байта.

В о с ь м е р и ч н ы й   ф о р м а т. Восьмеричный формат допускает восьмеричные цифры от 0 до 7 и обозначается последней буквой Q или O, например, 253Q. На сегодня восьмеричный формат используется весьма редко.

Д е с я т и ч н ы й   ф о р м а т   с   п л а в а ю щ е й   т о ч к о й. Этот формат поддерживается только ассемблером MASM.

При записи символьных и числовых констант следует помнить, что, например, символьная константа, определенная как DB '12', представляет символы ASCII и генерирует шест.3132, а числовая константа, определенная как DB 12, представляет двоичное число и генерирует шест.0C.

Рис.5.1 иллюстрирует директивы для определения различных символьных строк и числовых констант. Сегмент данных был ассемблирован для того, чтобы показать сгенерированный объектный код (слева).

ДИРЕКТИВА ОПРЕДЕЛЕНИЯ БАЙТА (DB)

Из различных директив, определяющих элементы данных, наиболее полезной является DB (определить байт). Символьное выражение в директиве DB может содержать строку символов любой длины, вплоть до конца строки (см. FLD2DB и FLD7DB на рис.5.1). Обратите внимание, что константа FLD2DB содержит символьную строку 'Personal Computer'. Объектный код показывает символы кода ASCII для каждого байта. Шест.20 представляет символ пробела.

Числовое выражение в директиве DB может содержать одну или более однобайтовых констант. Один байт выражается двумя шест. цифрами. Наибольшее положительное шест. число в одном байте это 7F, все "большие" числа от 80 до FF представляют отрицательные значения. В десятичном исчислении эти пределы выражаются числами +127 и -128.

В примере на рис.5.1 числовыми константами являются FLD3DB, FLD4DB, FLD5DB и FLD8DB. Поле FLD6DB представляет смесь из числовых и строковых констант, используемых для построения таблицы.

ДИРЕКТИВА ОПРЕДЕЛЕНИЯ СЛОВА (DW)

Директива DW определяет элементы, которые имеют длину в одно слово (два байта). Символьное выражение в DW ограничено двумя символами, которые ассемблер представляет в объектном коде так, что, например, 'PC' становится 'CP'. Для определения символьных строк директива DW имеет ограниченное применение.

Числовое выражение в DW может содержать одно или более двухбайтовых констант. Два байта представляются четырьмя шест. цифрами. Наибольшее положительное шест. число в двух байтах это 7FFF; все "большие" числа от 8000 до FFFF представляют отрицательные значения. В десятичном исчислении эти пределы выражаются числами +32767 и -32768.

В примере на рис.5.1 поля FLD1DW и FLD2DW определяют числовые константы. Поле FLD3DW определяет адрес - в данном случае смещение на адрес FLD7DB. В результате генерируется объектный код 0021 (R обозначает перемещаемость). Проверяя выше по рисунку, видно, что относительный адрес поля FLD7DB действительно 0021.

Поле FLD4DW определяет таблицу из пяти числовых констант. Заметим, что объектный код для каждой константы имеет длину в одно слово (два байта).

Для форматов директив DW, DD и DQ ассемблер преобразует константы в шест. объектный код, но записывает его в обратной последовательности. Таким образом десятичное значение 12345 преобразуется в шест.3039, но записывается в объектном коде как 3930.

ДИРЕКТИВА ОПРЕДЕЛЕНИЯ ДВОЙНОГО СЛОВА (DD)

Директива DD определяет элементы, которые имеют длину в два слова (четыре байта). Числовое выражение может содержать одну или более констант, каждая из которых имеет максимум четыре байта (восемь шест. цифр). Наибольшее положительное шест. число в четырех байтах это 7FFFFFFF; все "большие" числа от 80000000 до FFFFFFFF представляют отрицательные значения. В десятичном исчислении эти пределы выражаются числами +2147483647 и -2147483648.

В примере на рис.5.1 поле FLD3DD определяет числовую константу. В поле FLD4DD генерируется разница между двумя адресами, в данном случае результатом является длина поля FLD2DB. Поле FLD5DD определяет две числовые константы.

Ассемблер преобразует все числовые константы в директиве DD в шест. представление, но записывает объектный код в обратной последовательности. Таким образом десятичное значение 12345 преобразуется в шест.00003039, но

записывается в объектном коде как 39300000.

Символьное выражение директивы DD ограничено двумя символами. Ассемблер преобразует символы и выравнивает их слева в четырехбайтовом двойном слове, как показано в поле FLD2DD в объектном коде.

ДИРЕКТИВА ОПРЕДЕЛЕНИЯ УЧЕТВЕРЕННОГО СЛОВА (DQ)

Директива DQ определяет элементы, имеющие длину четыре слова (восемь байт). Числовое выражение может содержать одну или более констант, каждая из которых имеет максимум восемь байт или 16 шест. цифр. Наибольшее положительное шест. число - это семерка и 15 цифр F. Для получения представления о величине этого числа, покажем, что шест. 1 и 15 нулей эквивалентен следующему десятичному числу:

1152921504606846976

В примере на рис. 5.1 поля FLD2DQ и FLD3DQ иллюстрируют числовые значения. Ассемблер преобразует все числовые константы в директиве DQ в шест. представление, но записывает объектный код в обратной последовательности, как и в директивах DD и DW.

Обработка ассемблером символьных строк в директиве DQ аналогично директивам DD и DW.

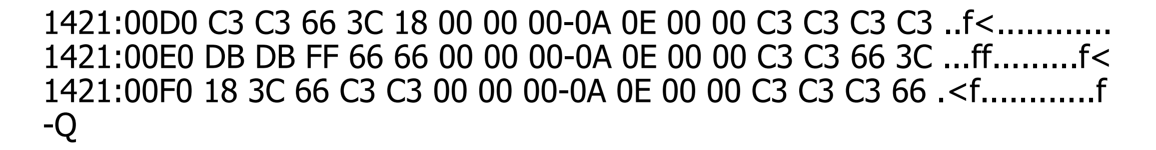
ДИРЕКТИВА ОПРЕДЕЛЕНИЯ ДЕСЯТИ БАЙТ (DT)

Директива DT определяет элементы данных, имеющие длину в десять байт. Назначение этой директивы связано с "упакованными десятичными" числовыми величинами (см. гл.13). По директиве DT генерируются различные константы, в зависимости от версии ассемблера; для практического применения ознакомьтесь с руководством по вашему ассемблера.

На рис.5.1 приведены примеры директивы DT для неопределенного элемента и для двухсимвольной константы.

Программа на рис.5.1 содержит только сегмент данных. Хотя ассемблер не выдает сообщений об ошибках, в таблице LINK MAP появится предупреждение: "Warning: No STACK Segment" (Предупреждение: не определен сегмент стека), а компоновщик LINK выдаст "There were 1 errors detected" (Обнаружена 1 ошибка). Несмотря на это предупреждение можно использовать отладчик DEBUG для просмотра объектного кода, как показано на рис.5.2.

```
D:\ D>DEBUG D:EXDEF.EXE
-D
1421:0000 00 50 65 72 73 6F 6E 61-6C 20 43 6F 6D 70 75 74 .Personal Comput
1421:0010 65 72 20 20 59 01 4A 41-4E 02 46 45 42 03 4D 41 .er Y.JAN.FEB.MA
1421:0020 52 33 32 36 35 34 00 00-00 00 00 00 00 00 00 00 R32654..........
1421:0030 F0 FF 59 00 21 00 03 00-04 00 07 00 08 00 09 00 ..Y.!...........
1421:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 43 50 .............CP
1421:0050 00 00 3C 7F 00 00 11 00-00 00 0E 00 00 00 31 00 ..<...........1.
1421:0060 00 00 00 00 00 00 00 00-00 00 47 4D 00 00 00 00 ..........GM....
1421:0070 00 00 3C 7F 00 00 00 00-00 00 00 00 00 00 00 00 ..<.............
-D
1421:0080 00 00 00 00 43 50 00 00-00 00 00 00 00 00 33 33 ....CP........33
1421:0090 3E 36 33 33 73 00 00 00-0A 0E 00 00 3E 63 63 30 >633s.......>cc0
1421:00A0 1C 06 63 63 3E 00 00 00-0A 0E 00 00 FF DB 99 18 ..cc>...........
1421:00B0 18 18 18 18 3C 00 00 00-0A 0E 00 00 63 63 63 63 ....<.......cccc
1421:00C0 63 63 63 63 3E 00 00 00-0A 0E 00 00 C3 C3 C3 C3 cccc>...........
```



```
1421:00D0 C3 C3 66 3C 18 00 00 00-0A 0E 00 00 C3 C3 C3 C3 ..f<...........
1421:00E0 DB DB FF 66 66 00 00 00-0A 0E 00 00 C3 C3 66 3C ...ff.........f<
1421:00F0 18 3C 66 C3 C3 00 00 00-0A 0E 00 00 C3 C3 C3 66 .<f...........f
-Q
```

Рис.5.2. Дамп сегмента данных.

Правая сторона дампа отчетливо показывает символьные данные, как, например, "Personal Computer".

НЕПОСРЕДСТВЕННЫЕ ОПЕРАНДЫ

На рис. 2.1 в главе 2 было показано использование непосредственных операндов. Команда

```
MOV   AX,0123H
```

пересылает непосредственную шест. константу 0123 в регистр AX. Трехбайтный объектный код для этой команды есть B82301, где B8 обозначает "переслать непосредственное значение в регистр AX", а следующие два байта содержат само значение. Многие команды имеют два операнда: первый может быть регистр или адрес памяти, а второй - непосредственная константа.

Использование непосредственного операнда более эффективно, чем определение числовой константы в сегменте данных и организация ссылки на нее в операнде команды MOV, например,

```
Сегмент данных:     AMT1 DW   0123H
Сегмент кодов:           MOV  AX,AMT1
```

Длина непосредственных операндов

Длина непосредственной константы зависит от длины первого операнда. Например, следующий непосредственный операнд является двухбайтовым, но регистр AL имеет только один байт:

```
MOV  AL,0123H  (ошибка)
```

однако, если непосредственный операнд короче, чем получающий операнд, как в следующем примере

```
ADD  AX,25H    (нет ошибки)
```

то ассемблер расширяет непосредственный операнд до двух байт, 0025 и записывает объектный код в виде 2500.

Непосредственные форматы

Непосредственная константа может быть шестнадцатиричной, например, 0123H; десятичной, например, 291 (которую ассемблер конвертирует в шест.0123); или двоичной, например, 100100011B (которая преобразуется в шест. 0123).

Ниже приведен список команд, которые допускают непосредственные операнды:

Команды пересылки и сравнения: MOV, CMP.
Арифметические команды: ADC, ADD, SBB, SUB.
Команды сдвига: RCL, RCR, ROL, ROR, SHL, SAR, SHR.
Логические команды: AND, OR, TEST, XOR.

На рис.5.3 приведены примеры допустимых команд с непосредственными операндами. В последующих главах будут объяснены команды арифметического переноса, сдвига и логические команды. Поскольку сейчас данные примеры не предназначены для выполнения, в них опущено определение стека и инициализация сегментных регистров.

```
                         page    60,132
                  TITLE   EXIMM   (EXE) Пример непосредств. операндов
                  ;       (Кодируется для ассемблирования,
                  ;               но не для выполнения)
0000                DATASG  SEGMENT PARA 'Data'
0000 ??              FLD1    DB      ?
0001 ????             FLD2    DW      ?
0003                DATASG  ENDS

0000                CODESEG  SEGMENT PARA 'Code'
0000                BEGIN   PROC    FAR
                        ASSUME  CS:CODESG,DS:DATASG

                  ;            Операции пересылки и сравнения:
                  ;            ------------------------------
0000 BB 0113               MOV     BX,275   ;Пересылка
0003 3C 19                CMP     AL,H     ;Сравнение

                  ;            Арифметические операции:
                  ;            -----------------------
0005 14 05                ADC     AL,5     ;Сложение с переносом
0007 80 C7 0C              ADD     BH,12    ;Сложение
000A 1C 05                SBB     AL,5     ;Вычитание с заемом
000C 80 2E 000 R 05          SUB     FLD1,5   ;Вычитание

                  ;            Ротация и сдвиг (только на 1 бит):
                  ;            ---------------------------------
0011 D0 D3                RCL     BL,1     ;Ротация влево с переносом
0013 D0 DC                RCR     AH,1     ;Ротация вправо с переносом
0015 D1 06 0001 R            ROL     FID2,1   ;Ротация влево
0019 D0 C8                ROR     AL,1     ;Ротация вправо
001B D1 E1                SAL     CX,1     ;Сдвиг влево
001D D1 FB                SAR     BX,1     ;Арифм. сдвиг вправо
001F D0 2E 0000 R            SHR     FLD1,1   ;Сдвиг вправо

                  ;            Логические операции:
                  ;            -------------------
0023 24 2C                AND     AL,00101100B ;AND (регистр)
0025 80 CF 2A              OR      BH,2AH   ;OR (регистр)
0028 F6 C3 7A              TEST    BL,7AH   ;TEST (регистр)
002B 80 36 0000 R 23         OR      FLD1,23H ;XOR (память)
0030                BEGIN   ENDP
0030                CODESG  ENDS
                        END
```

Рис.5.3. Команды с непосредственными данными.

Для создания элементов, длиннее чем два байта, можно использовать цикл (см. гл.7) или строковые команды (см. гл.11).

ДИРЕКТИВА EQU

Директива EQU не определяет элемент данных, но определяет значение, которое может быть использовано для постановки в других командах. Предположим, что в сегменте данных закодирована следующая директива EQU:

```
TIMES     EQU  10
```

Имя, в данном случае TIMES, может быть представлено любым допустимым в ассемблере именем. Теперь, в какой-бы команде или директиве не использовалось слово TIMES ассемблер подставит значение 10. Например, ассемблер преобразует директиву

```
FIELDA    DB   TIMES DUP (?)
```

в

```
FIELDA    DB   10 DUP (?)
```

Имя, связанное с некоторым значением с помощью директивы EQU, может использоваться в командах, например:

```
COUNTR    EQU  05
          ...
          MOV  CX,COUNTR
```

Ассемблер заменяет имя COUNTR в команде MOV на значение 05, создавая операнд с непосредственным значением, как если бы было закодировано

```
MOV  CX,05     ;Ассемблер подставляет 05
```

Здесь преимущество директивы EQU заключается в том, что многие команды могут использовать значение, определенное по имени COUNTR. Если это значение должно быть изменено, то изменению подлежит лишь одна директива EQU. Естественно, что использование директивы EQU разумно лишь там, где подстановка имеет смысл для ассемблера. В директиве EQU можно использовать символические имена:

```
1.   TP   EQU  TOTALPAY
2.   MPY  EQU  MUL
```

Первый пример предполагает, что в сегменте данных программы определено имя TOTALPAY. Для любой команды, содержащей операнд TP, ассемблер заменит его на адрес TOTALPAY. Второй пример показывает возможность использования в программе слова MPY вместо обычного мнемокода MUL.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Имена элементов данных в программе должны быть уникальны и по возможности наглядны. Например, элемент для зарплаты служащего может иметь имя EMPWAGE.

- Для определения символьных строк используйте директиву DB, так как ее формат допускает строки длиннее двух байт и формирует их в нормальной последовательности (слева-направо).

- Будьте внимательны при указании десятичных и шест. значений. Сравните, например, сложение содержимого регистра AX с десятичным 25 и с шест. 25:

```
ADD  AX,25     ;Прибавить 25
ADD  AX,25H    ;Прибавить 37
```

- Помните, что директивы DW, DD и DQ записывают числовое значение в объектном коде в обратной последовательности байт.

- Используйте элементы DB для операций с полурегистрами (AL, AH, BL и т.д.) и DW для операций с полными регистрами (AX, BX, CX и т.д.). Числовые элементы, определенные директивами DD и DQ имеют специальное применение.

- Следите за соответствием непосредственных операндов размеру регистра: однобайтовая константа - однобайтовый регистр (AL, BH), двухбайтовая константа - полный регистр (AX, BX).

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

5.1. Какова длина в байтах для элементов данных, определенных директивами: а) DW, б) DD, в) DT, г) DB, д) DQ?

5.2. Определите символьную строку по имени TITLE1, содержащую константу RGB Electronics.

5.3. Определите следующие числовые значения в элементах данных с именами от FLDA до FLDE:

a) четырехбайтовый элемент, содержащий шест. эквивалент десятичного числа 115;
b) однобайтовый элемент, содержащий шест. эквивалент десятичного числа 25;
c) двухбайтовый элемент, содержащий неопределенное значение;
d) однобайтовый элемент, содержащий двоичной эквивалент десятичного числа 25;
e) директиву DW, содержащую последовательные значения 16, 19, 20, 27, 30.

5.4. Покажите сгенерированный шест. объектный код для а) DB '26' и б) DB 26.

5.5. Определите ассемблерный шест. объектный код для а) DB 26H, б) DW 2645H, в) DD 25733AH, г) DQ 25733AH.

5.6. Закодируйте следующие команды с непосредственными операндами:

а) загрузить 320 в регистр AX;
б) сравнить поле FLDB с нулем;
в) прибавить шест. 40 к содержимому регистра BX;
г) вычесть шест. 40 из регистра CX;
д) сдвинуть содержимое поля FLDB на один бит влево;
е) сдвинуть содержимое регистра CH на один бит вправо.

5.7. Введите и ассемблируйте элементы данных и команды из вопросов 5.2, 5.3 и 5.6. Стек для этого упражнения не требуется. Также не следует выполнять компоновку. Для проверки ассемблированного кода используйте отладчик DEBUG. Распечатайте LST-файл (листинг), если в результате

ассемблирования не будет сообщений об ошибках. Не забудьте команду MODE LPT1:132,6 для установки ширины печати.

ГЛАВА 6 Программы в COM-файлах

Ц е л ь: Объяснить назначение и использование COM-файлов и перевод ассемблерных программ в формат COM-файлов.

ВВЕДЕНИЕ

До сих пор вы писали, ассемблировали и выполняли програм мы в EXE-формате. Компановщик LINK автоматически генерирует особый формат для EXE-файлов, в котором присутствует специальный начальный блок (заголовок) размером не менее 512 байт. (В гл.22 рассматривается содержимое начальных блоков).

Для выполнения можно также создавать COM-файлы. Примером часто используемого COM-файла является COMMAND.COM. Программа EXE2BIN.COM в оперативной системе DOS преобразует EXE-файлы в COM-файлы. Фактически эта программа создает BIN (двоичный) файл, поэтому она и называется "преобразователь EXE в Bin (EXE-to-BIN)". Выходной Bin-файл можно переименовать в COM-файл.

РАЗЛИЧИЯ МЕЖДУ ПРОГРАММАМИ В EXE и COM-файлах

Несмотря на то, что EXE2BIN преобразует EXE-файл в COM-файл, существуют определенные различия между программой, выполняемой как EXE-файл и программой, выполняемой как COM-файл.

Р а з м е р п р о г р а м м ы. EXE-программа может иметь любой размер, в то время как COM-файл ограничен размером одного сегмента и не превышает 64К. COM-файл всегда меньше, чем соответствующий EXE-файл; одна из причин этого - отсутствие в COM-файле 512-байтового начального блока EXE-файла.

С е г м е н т с т е к а. В EXE-программе определяется сегмент стека, в то время как COM-программа генерирует стек автоматически. Таким образом при создании ассемблерной программы, которая будет преобразована в COM-файл, стек должен быть опущен.

С е г м е н т д а н н ы х. В EXE программе обычно определяется сегмент данных, а регистр DS инициализируется адресом этого сегмента. В COM-программе все данные должны быть определены в сегменте кода. Ниже будет показан простой способ решения этого вопроса.

И н и ц и а л и з а ц и я. EXE-программа записывает нулевое слово в стек и инициализирует регистр DS. Так как COM-программа не имеет ни стека, ни сегмента данных, то эти шаги отсутствуют. Когда COM-программа начинает работать, все сегментные регистры содержат адрес префикса программного сегмента (PSP), - 256-байтового (шест. 100) блока, который резервируется операционной системой DOS непосредственно перед COM или EXE программой в памяти. Так как адресация начинается с шест. смещения 100 от начала PSP, то в программе после оператора SEGMENT кодируется директива ORG 100H.

О б р а б о т к а. Для программ в EXE и COM форматах выполняется

ассемблирование для получения OBJ-файла, и компановка для получения EXE-файла. Если программа создается для выполнения как EXE-файл, то ее уже можно выполнить. Если же программа создается для выполнения как COM-файл, то компановщиком будет выдано сообщение:

Warning: No STACK Segment
(Предупреждение: сегмент стека не определен)

Это сообщение можно игнорировать, так как определение стека в программе не предполагалось. Для преобразования EXE-файла в COM-файл используется программа EXE2BIN. Предположим, что EXE2BIN имеется на дисководе A, а скомпонованный файл по имени CALC.EXE - на дисководе B. Введите

```
EXE2BIN B:CALC,B:CALC.COM
```

Так как первый операнд всегда предполагает EXE файл, то можно не кодировать тип EXE. Второй операнд может иметь другое имя (не CALC.COM). Если не указывать тип COM, то EXE2BIN примет по умолчанию тип BIN, который впоследствии можно переименовать в COM. После того как преобразование будет выполнено можно удалить OBJ- и EXE-файлы.

Если исходная программа написана для EXE-формата, то можно, используя редактор, заменить команды в исходном тексте для COM файла.

ПРИМЕР COM-ПРОГРАММЫ

Программа EXCOM1, приведенная на рис.6.1, аналогична программе на рис.4.3, но изменена согласно требований COM-формата. Обратите внимание на следующие изменения в этой COM-программе:

- Стек и сегмент данных отсутствует.
- Оператор ASSUME указывает ассемблеру установить относительные адреса с начала сегмента кодов. Регистр CS также содержит этот адрес, являющийся к тому же адресом префикса программного сегмента (PSP). Директива ORG служит для резервирования 100 (шест.) байт от начального адреса под PSP.
- Директива ORG 100H устанавливает относительный адрес для начала выполнения программы. Программный загрузчик использует этот адрес для командного указателя.
- Команда JMP используется для обхода данных, определенных в программе.

Ниже показаны шаги для обработки и выполнения этой программы:

```
MASM [ответы на запросы обычные]
LINK [ответы на запросы обычные]
EXE2BIN B:EXCOM1,B:EXCOM1.COM
DEL B:EXCOM1.OBJ,B:EXCOM1.EXE (удаление OBJ и EXE-файлов)
```

Размеры EXE- и COM-программ - 788 и 20 байт соответственно. Учитывая такую эффективность COM-файлов, рекомендуется все небольшие программы создавать для COM-формата. Для трассировки выполнения программы от начала (но не включая) команды RET введите DEBUG B:EXCOM1.COM.

Некоторые программисты кодируют элементы данных после команд так, что первая команда JMP не требуется. Кодирование элементов данных перед командами позволяет ускорить процесс ассемблирования и является методикой, рекомендуемой в руководстве по ассемблеру.

```
       page 60,132
TITLE   XCOM1  COM-программа для пересылки и сложения
CODESG  SEGMENT PARA 'Code'
       ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:CODESG
       ORG     100H            ;Начало в конце PSP
BEGIN:  JMP     MAIN            ;Обход через данные
; -------------------------------------------------
FLDA    DW      250             ;Определение данных
FLDB    DW      125
FLDC    DW      ?
; -------------------------------------------------
MAIN    PROC    NEAR
       MOV     AX,FLDA         ;Переслать 0250 в AX
       ADD     AX,FLDB         ;Прибавить 0125 к AX
       MOV     FLDC,AX         ;Записать сумму в FLDC
       RET                     ;Вернуться в DOS
MAIN    ENDP
CODESG  ENDS
       END     BEGIN
```

Рис.6.1. Пример COM-программы.

СТЕК ДЛЯ COM-ПРОГРАММЫ

Для COM-файла DOS автоматически определяет стек и устанавливает одинаковый общий сегментный адрес во всех четырех сегментных регистрах. Если для программы размер сегмента в 64К является достаточным, то DOS устанавливает в регистре SP адрес конца сегмента - шест.FFFE. Это будет верх стека. Если 64К байтовый сегмент не имеет достаточно места для стека, то DOS устанавливает стек в конце памяти. В обоих случаях DOS записывает затем в стек нулевое слово.

Возможность использования стека зависит от размера программы и ограниченности памяти. С помощью команды DIR можно определить размер файла и вычислить необходимое пространство для стека.

Все небольшие программы в этой книге в основном расчитаны на COM-формат.

ОСОБЕННОСТЬ ОТЛАДКИ

Несоблюдение хотя бы одного требования COM-формата может послужить причиной неправильной работы программы. Если EXE2BIN обнаруживает ошибку, то выдается сообщение о невозможности преобразования файла без указания конкретной причины. Необходимо проверить в этом случае директивы SEGMENT, ASSUME и END. Если опущен ORG 100H, то на данные в префиксе программного сегмента будут установлены неправильные ссылки с непредсказуемым результатом при выполнении.

При выполнении COM-программы под управлением отладчика DEBUG необходимо использовать команду D CS:100 для просмотра данных и команд. Не следует выполнять в отладчике команду RET; предпочтительнее использовать команду Q отладчика. Некоторые программисты используют INT 20H вместо команды RET.

Попытка выполнить EXE-модуль программы, написанной для COM-формата, не имеет успеха.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Объем COM-файла ограничен 64K.

- COM-файл меньше, чем соответствующий EXE-файл.

- Программа, написанная для выполнения в COM-формате не содержит стека и сегмента данных и не требует инициализации регистра DS.

- Программа, написанная для выполнения в COM-формате использует директиву ORG 100H после директивы SEGMENT для выполнения с адреса после префикса программного сегмента.

- Программа EXE2BIN преобразует EXE-файл в COM-файл, обусловленный указанием типа COM во втором операнде.

- Операционная система DOS определяет стек для COM-программы или в конце программы, если позволяет размер, или в конце памяти.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

6.1. Каков максимальный размер COM-файла?

6.2. Какие сегменты можно определить в программе, которая будет преобразована в COM-файл?

6.3. Как обходится COM-файл при выполнении с фактом отсутствия определения стека?

6.4. Программа в результате компановки получала имя SAMPLE.EXE. Напишите команду DOS для преобразования ее в COM-файл.

6.5. Измените программу из вопроса 4.6 для COM-формата, обработайте ее и выполните под управлением отладчика DEBUG.

ГЛАВА 7 Логика и Организация Программы

Ц е л ь: Раскрыть механизм передачи управления в программе (циклы и переходы) для логических сравнений и программной организации.

ВВЕДЕНИЕ

До этой главы примеры выполнялись последовательно команда за командой. Однако, программируемые задачи редко бывают так просты. Большинство программ содержат ряд циклов, в которых несколько команд повторяются до достижения определенного требования, и различные проверки, определяющие, какие из нескольких действий следует выполнять. Обычным требованием является проверка - должна ли программа завершить выполнение.

Эти требования включают передачу управления по адресу команды, которая не находится непосредственно за выполняемой в текущий момент командой. Передача управления может осуществляться вперед для выполнения новой группы команд или назад для повторения уже выполненных команд.

Некоторые команды могут передавать управление, изменяя нормальную

последовательность шагов непосредственной модификацией значения смещения в командном указателе. Ниже приведены четыре способа передачи управления (все будут рассмотрены в этой главе):

Безусловный переход: JMP
Цикл: LOOP
Условный переход: Jnnn (больше,меньше,равно)
Вызов процедуры: CALL

Заметим, что имеется три типа адресов: SHORT, NEAR и FAR. Адресация SHORT используется при циклах, условных переходах и некоторых безусловных переходах. Адресация NEAR и FAR используется для вызовов процедур (CALL) и безусловных переходов, которые не квалифицируются , как SHORT. Все три типа передачи управления воздействуют на содержимое регистра IP; тип FAR также изменяет регистр CS.

КОМАНДА JMP

Одной из команд обычно используемых для передачи управления является команда JMP. Эта команда выполняет безусловный переход, т.е. обеспечивает передачу управления при любых обстоятельствах.
В COM-программе на рис. 7.1 используется команда JMP. В регистры AX, BX, и CX загружается значение 1, и затем в цикле выполняются следующие операции:

прибавить 1 к регистру AX,
прибавить AX к BX,
удвоить значение в регистре CX.

Повторение цикла приводит к увеличению содержимого регистра AX: 1,2,3,4..., регистра BX: 1,3,6,10..., и регистра CX: 1,2,4,8... Начало цикла имеет метку, в данном случае, A20: - двоеточие обозначает, что метка находится внутри процедуры (в данном случае BEGIN) в сегменте кода. В конце цикла находится команда

JMP A20

которая указывает на то, что управление должно быть передано команде с меткой A20. Обратите внимание, что адресная метка в операнде команды указывается без двоеточия. Данный цикл не имеет выхода и приводит к бесконечному выполнению - такие циклы обычно не используются.

```
                    page 60,132
              TITLE   EXJUMP  (COM) Организация цикла с помощью JMP
0000            CODESG  SEGMENT PARA 'Code'
                    ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
0100                  ORG     100H

0100            MAIN    PROC    NEAR
0100  B8 0001           MOV     AX,01          ;Инициализация AX,
0103  BB 0001           MOV     BX,01          ;              BX,
0106  B9 0001           MOV     CX,01          ;            и CX
0109            A20:
0109  05 0001           ADD     AX,01          ;Прибавить 01 к AX
010C  03 D8             ADD     BX,AX          ;Прибавить 01 к BX
010E  D1 E1             SHL     CX,1           ;Удвоить CX
0110  EB F7             JMP     A20            ;Переход на A20
```

```
0112           MAIN    ENDP
0112           CODESG  ENDS
                   END     MAIN
```

Рис.7.1. Использование команды JMP.

Метку можно кодировать на одной строке с командой:

```
A20:   ADD   AX,01
```

или на отдельной строке:

```
A20:
     ADD   AX,01
```

В обоих случаях адрес A20 указывает на первый байт команды ADD. Двоеточие в метке A20 указывает на тип метки - NEAR. Запомните: отсутствие двоеточия в метке является частой ошибкой. В нашем примере A20 соответствует -9 байтам от команды JMP, в чем можно убедиться по объектному коду команды - EBF7. EB представляет собой машинный код для короткого перехода JMP, а F7 - отрицательное значение смещения (-9). Команда JMP прибавляет F7 к командному указателю (IP), который содержит адрес команды после JMP (0112):

```
                         Дес. Шест.
Командный указатель:     274  112
Адрес в команде JMP:      -9   F7 (двоичное дополнение)
                         ---  ---
Адрес перехода:          265  109
```

В результате сложения получается адрес перехода - шест.109. Проверьте по листингу программы, что относительный адрес метки действительно соответствует шест.109. Соответственно операнд в команде JMP для перехода вперед имеет положительное значение.

Команда JMP для перехода в пределах -128 до +127 байт имеет тип SHORT. Ассемблер генерирует в этом случае однобайтовый операнд в пределах от 00 до FF. Команда JMP, превосходящая эти пределы, получает тип FAR, для которого генерируется другой машинный код и двухбайтовый операнд. Ассемблер в первом просмотре исходной программы определяет длину каждой команды. Однако, команда JMP может быть длиной два или три байта. Если к моменту просмотра команды JMP ассемблер уже вычислил значение операнда (при переходе назад):

```
A50:
    ...
    JMP  A50
```

то он генерирует двухбайтовую команду. Если ассемблер еще не вычислил значение операнда (при переходе вперед)

```
    JMP  A90
    ...
A90:
```

то он не знает тип перехода NEAR или FAR, и автоматически генерирует 3-х байтовую команду. Для того, чтобы указать ассемблеру на необходимость генерации двухбайтовой команды, следует использовать оператор SHORT:

```
    JMP  SHORT A90
```

```
   ...
A90:
```

В качестве полезного упражнения, введите программу, проассемблируйте ее, скомпонуйте и переведите в COM-формат. Определение данных не требуется, поскольку непосредственные операнды генерируют все необходимые данные. Используйте отладчик DEBUG для пошагового выполнения COM-модуля и просмотрите несколько повторений цикла. Когда регистр AX будет содержать 08, BX и CX увеличатся до шест.24 (дес.36) и шест.80 (дес.128), соответственно. Для выхода из отладчика используйте команду Q.

КОМАНДА LOOP

Команда JMP в примере на рис.7.1 реализует бесконечный цикл. Но более вероятно подпрограмма должна выполнять определенное число циклов. Команда LOOP, которая служит для этой цели, использует начальное значение в регистре CX. В каждом цикле команда LOOP автоматически уменьшает содержимое регистра CX на 1. Пока значение в CX не равно нулю, управление передается по адресу, указанному в операнде, и если в CX будет 0, управление переходит на следующую после LOOP команду.

```
                  page 60,132
           TITLE   EXLOOP  (COM) Организация цикла командой LOOP
0000         CODESG  SEGMENT PARA 'Code'
                  ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
0100               ORG     100H

0100         BEGIN   PROC    NEAR
0100  B8 0001          MOV     AX,01      ;Инициализация AX,
0103  BB 0001          MOV     BX,01      ;               BX,
0106  BA 0001          MOV     DX,01      ;             и DX
0109  B9 000A          MOV     CX,10      ;Число циклов
010C         A20:
010C  40             INC     AX         ;Прибавить 01 к AX
010D  03 D8            ADD     BX,AX      ;Прибавить AX к BX
010F  D1 E2            SHL     DX,1       ;Удвоить DX
0111  E2 F9            LOOP    A20        ;Уменьшить CX и повторить
                                 ; цикл, если не нуль
0113  C3             RET               ;Завершить работу
0114         BEGIN   ENDP
0114         CODESG  ENDS
                  END     BEGIN
```

Рис.7.2. Использование команды LOOP.

Программа на рис.7.2, иллюстрирующая использование команды LOOP, выполняет действия, аналогичные примеру на рис.7.1 за исключением того, что после десяти циклов программа завершается. Команда MOV инициализирует регистр CX значением 10. Так как команда LOOP использует регистр CX, то в программе для удвоения начального значения 1 вместо регистра CX используется DX. Команда JMP A20 заменена командой LOOP и для эффективности команда ADD AX,01 заменена командой INC AX (увеличение AX на 1).

Аналогично команде JMP, операнд команды LOOP определяет расстояние от конца команды LOOP до адреса метки A20, которое прибавляется к содержимому

командного указателя. Для команды LOOP это расстояние должно быть в пределах от -128 до +127 байт. Если операнд превышает эти границы, то ассемблер выдаст сообщение "Relative jump out of range" (превышены границы перехода).

Для проверки команды LOOP рекомендуется изменить соответствующим образом программу, приведенную на рис.7.1, выполнить ее ассемблирование, компоновку и преобразование в COM-файл. Для трассировки всех десяти циклов используйте отладчик DEBUG. Когда в значение регистре CX уменьшится до нуля, содержимое регистров AX, BX и DX будет соответственно шест. 000B, 0042 и 0400. Для выхода из отладчика введите команду Q.

Дополнительно существует две разновидности команды LOOP - это LOOPE (или LOOPZ) и LOOPNE (или LOOPNZ). Обе команды также уменьшают значение регистра CX на 1. Команда LOOPE передает управление по адресу операнда, если регистр CX имеет ненулевое значение и флаг нуля установлен (ZF=1). Команда LOOPNE передает управление по адресу операнда, если регистр CX имеет ненулевое значение и флаг нуля сброшен (ZF=0).

ФЛАГОВЫЙ РЕГИСТР

Следующий материал данной главы требует более детального ознакомления с флаговым регистром. Этот регистр содержит 16 бит флагов, которые управляются различными командами для индикации состояния операции. Во всех случаях флаги сохраняют свое значение до тех пор, пока другая команда не изменит его. Флаговый регистр содержит следующие девять используемых бит (звездочками отмечены неиспользуемые биты):

```
Номер бита:    15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Флаг:           *  *  *  *  O  D I T S Z * A * P * C
```

Рассмотрим эти флаги в последовательности справа налево.

CF (Carry Flag) - флаг переноса. Содержит значение "переносов" (0 или 1) из старшего разряда при арифметических операциях и некоторых операциях сдвига и циклического сдвига (см. гл.12).

PF (Parity Flag) - флаг четности. Проверяет младшие восемь бит результатов операций над данными. Нечетное число бит приводит к установке этого флага в 0, а четное - в 1. Не следует путать флаг четности с битом контроля на четность.

AF (Auxiliary Carry Flag) - дополнительный флаг переноса. Устанавливается в 1, если арифметическая операция приводит к переносу четвертого справа бита (бит номер 3) в регистровой однобайтовой команде. Данный флаг имеет отношение к арифметическим операциям над символами кода ASCII и к десятичным упакованным полям.

ZF (Zero Flag) - флаг нуля. Устанавливается в качестве результата арифметических команд и команд сравнения. Как это ни странно, ненулевой результат приводит к установке нулевого значения этого флага, а нулевой - к установке единичного значения. Кажущееся несоответствие является, однако, логически правильным, так как 0 обозначает "нет" (т.е. результат не равен нулю), а единица обозначает "да" (т.е. результат равен нулю). Команды условного перехода JE и JZ проверяют этот флаг.

SF (SIgn Flag) - знаковый флаг. Устанавливается в соответствии со знаком результата (старшего бита) после арифметических операций: положительный результат устанавливает 0, а отрицательный - 1. Команды условного перехода JG и JL проверяют этот флаг.

TF (Trap Flag) - флаг пошагового выполнения. Этот флаг вам уже приходилось устанавливать, когда использовалась команда T в отладчике DEBUG. Если этот флаг установлен в единичное состояние, то процессор переходит в режим пошагового выполнения команд, т.е. в каждый момент выполняется одна команда под пользовательским управлением.

IF (Interrupt Flag) - флаг прерывания. При нулевом состоянии этого флага прерывания запрещены, при единичном - разрешены.

DF (DIrection Flag) - флаг направления. Используется в строковых операциях для определения направления передачи данных. При нулевом состоянии команда увеличивает содержимое регистров SI и DI, вызывая передачу данных слева направо, при нулевом - уменьшает содержимое этих регистров, вызывая передачу данных справа налево (см. гл.11).

OF (Overflow Flag) - флаг переполнения. Фиксирует арифметическое переполнение, т.е. перенос вниз старшего (знакового) бита при знаковых арифметических операциях.

В качестве примера: команда CMP сравнивает два операнда и воздействует на флаги AF, CF, OF, PF, SF, ZF. Однако, нет необходимости проверять все эти флаги по отдельности. В следующем примере проверяется содержит ли регистр BX нулевое значение:

```
                CMP  BX,00        ;Сравнение BX с нулем
                JZ   B50          ;Переход на B50 если нуль
                .    (действия при не нуле)
                .
        B50:     ...              ;Точка перехода при BX=0
```

Если BX содержит нулевое значение, команда CMP устанавливает флаг нуля ZF в единичное состояние, и возможно изменяет (или нет) другие флаги. Команда JZ (переход, если нуль) проверяет только флаг ZF. При единичном значении ZF, обозначающее нулевой признак, команда передает управление на адрес, указанный в ее операнде, т.е. на метку B50.

КОМАНДЫ УСЛОВНОГО ПЕРЕХОДА

В предыдущих примерах было показано, что команда LOOP уменьшает на единицу содержимое регистра CX и проверяет его: если не ноль, то управление передается по адресу, указанному в операнде. Таким образом, передача управления зависит от конкретного состояния. Ассемблер поддерживает большое количество команд условного перехода, которые осуществляют передачу управления в зависимости от состояний флагового регистра. Например, при сравнении содержимого двух полей последующий переход зависит от значения флага.

Команду LOOP в программе на рис.7.2 можно заменить на две команды: одна уменьшает содержимое регистра CX, а другая выполняет условный переход:

```
   Использование LOOP    Использование условного перехода

        LOOP A20               DEC  CX
                               JNZ  A20
```

Команды DEC и JNZ действуют аналогично команде LOOP: уменьшают содержимое регистра CX на 1 и выполняет переход на метку A20, если в CX не ноль. Команда DEC кроме того устанавливает флаг нуля во флаговом регистре в состояние 0 или 1. Команда JNZ затем проверяет эту установку. В

рассмотренном примере команда LOOP хотя и имеет ограниченное использование, но более эффективна, чем две команды: DEC и JNZ.

Аналогично командам JMP и LOOP операнд в команде JNZ содержит значение расстояния между концом команды JNZ и адресом A20, которое прибавляется к командному указателю. Это расстояние должно быть в пределах от -128 до +127 байт. В случае перехода за эти границы ассемблер выдаст сообщение "Relative jump out of range" (превышены относительные границы перехода).

Знаковые и беззнаковые данные

Рассматривая назначение команд условного перехода следует пояснить характер их использования. Типы данных, над которыми выполняются арифметические операции и операции сравнения определяют какими командами пользоваться: беззнаковыми или знаковыми. Беззнаковые данные используют все биты как биты данных; характерным примером являются символьные строки: имена, адреса и натуральные числа. В знаковых данных самый левый бит представляет собой знак, причем если его значение равно нулю, то число положительное, и если единице, то отрицательное. Многие числовые значения могут быть как положительными так и отрицательными.

В качестве примера предположим, что регистр AX содержит 11000110, а BX - 00010110. Команда

```
CMP  AX,BX
```

сравнивает содержимое регистров AX и BX. Если данные беззнаковые, то значение в AX больше, а если знаковые - то меньше.

Переходы для беззнаковых данных

Мнемоника	Описание	Проверяемые флаги
JE/JZ	Переход, если равно/нуль	ZF
JNE/JNZ	Переход, если не равно/не нуль	ZF
JA/JNBE	Переход, если выше/не ниже или равно	ZF,CF
JAE/JNB	Переход, если выше или равно/не ниже	CF
JB/JNAE	Переход, если ниже/не выше или равно	CF
JBE/JNA	Переход, если ниже или равно/не выше	CF,AF

Любую проверку можно кодировать одним из двух мнемонических кодов. Например, JB и JNAE генерирует один и тот же объектный код, хотя положительную проверку JB легче понять, чем отрицательную JNAE.

Переходы для знаковых данных

Мнемоника	Описание	Проверяемые флаги
JE/JZ	Переход, если равно/нуль	ZF
JNE/JNZ	Переход, если не равно/не нуль	ZF
JG/JNLE	Переход, если больше/не меньше или равно	ZF,SF,OF
JGE/JNL	Переход, если больше или равно/не меньше	SF,OF
JL/JNGE	Переход, если меньше/не больше или равно	SF,OF
JLE/JNG	Переход, если меньше или равно/не больше	ZF,SF,OF

Команды перехода для условия равно или ноль (JE/JZ) и не равно или не ноль (JNE/JNZ) присутствуют в обоих списках для беззнаковых и знаковых данных. Состояние равно/нуль происходит вне зависимости от наличия знака.

Специальные арифметические проверки

```
Мнемоника   Описание                        Проверяемые флаги

JS      Переход, если есть знак (отрицательно)       SF
JNS     Переход, если нет знака(положительно)        SF
JC      Переход, если есть перенос (аналогично JB)   CF
JNC     Переход, если нет переноса                   CF
JO      Переход, если есть переполнение              OF
JNO     Переход, если нет переполнения               OF
JP/JPE  Переход, если паритет четный                 PF
JNP/JP  Переход, если паритет нечетный               PF
```

Еще одна команда условного перехода проверяет равно ли содержимое регистра CX нулю. Эта команда необязательно должна располагаться непосредственно за командой арифметики или сравнения. Одним из мест для команды JCXZ может быть начало цикла, где она проверяет содержит ли регистр CX ненулевое значение.

Не спешите пока заучивать эти команды наизусть. Запомните только, что для беззнаковых данных есть переходы по состояниям равно, выше или ниже, а для беззнаковых - равно, больше или меньше. Переходы по проверкам флагов переноса, переполнения и паритета имеют особое назначение. Ассемблер транслирует мнемонические коды в объектный код независимо от того, какую из двух команд вы применили. Однако, команды JAE и JGE являясь явно одинаковыми, проверяют различные флаги.

ПРОЦЕДУРЫ И ОПЕРАТОР CALL

В предыдущих главах примеры содержали в кодовом сегменте только одну процедуру, оформленную следующим образом:

```
BEGIN   PROC FAR
    .
    .
BEGIN   ENDP
```

Операнд FAR информирует систему о том, что данный адрес является точкой входа для выполнения, а директива ENDP определяет конец процедуры. Кодовый сегмент, однако, может содержать любое количество процедур, которые разделяются директивами PROC и ENDP. Типичная организация многопроцедурной программы приведена на рис.7.3.

```
-----------------------¬
¦ CODESG  SEGMENT  PARA ¦
+----------------------+
¦ BEGIN   PROC    FAR   ¦
¦        .             ¦
¦        .             ¦
¦       CALL    B10    ¦
¦       CALL    C10    ¦
¦       RET            ¦
¦ BEGIN   ENDP          ¦
+----------------------+
¦ B10     PROC    NEAR  ¦
¦        .             ¦
¦        .             ¦
```

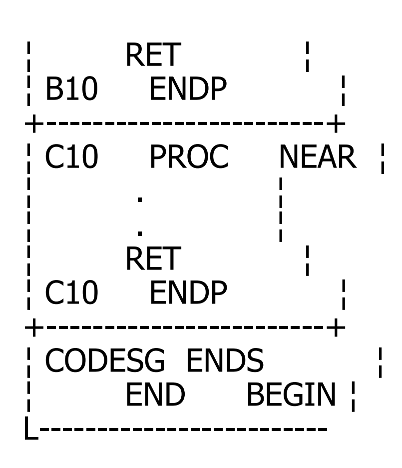


Рис.7.3. Вызов процедур.

Обратите внимание на следующие особенности:

- Директивы PROC по меткам B10 и C10 имеют операнд NEAR для указания того, что эти процедуры находятся в текущем кодовом сегменте. Во многих последующих примерах этот операнд опущен, так как по умолчанию ассемблер принимает тип NEAR.

- Каждая процедура имеет уникальное имя и содержит собственную директиву ENDP для указания конца процедуры.

- Для передачи управления в процедуре BEGIN имеются две команды: CALL B10 и CALL C10. В результате первой команды CALL управление передается процедуре B10 и начинается ее выполнение. Достигнув команды RET, управление возвращается на команду непосредственно следующую за CALL B10. Вторая команда CALL действует аналогично - передает управление в процедуру C10, выполняет ее команды и возвращает управление по команде RET.

- Команда RET всегда выполняет возврат в вызывающую программу. Программа BEGIN вызывает процедуры B10 и C10, которые возвращают управление обратно в BEGIN. Для выполнения самой программы BEGIN операционная система DOS вызывает ее и в конце выполнения команда RET возвращает управление в DOS. Если процедура B10 не содержит завершающей команды RET, то выполнение команд продолжится из B10 непосредственно в процедуре C10. Если процедура C10 не содержит команды RET, то будут выполняться команды, оказавшиеся за процедурой C10 с непредсказуемым результатом.

Использование процедур дает хорошую возможность организовать логическую структуру программы. Кроме того, операнды для команды CALL могут иметь значения, выходящие за границу от -128 до +127 байт.
Технически управление в процедуру типа NEAR может быть передано с помощью команд перехода или даже обычным построчным кодированием. Но в большинстве случаев рекомендуется использовать команду CALL для передачи управления в процедуру и команду RET для возврата.

СЕГМЕНТ СТЕКА

До этого раздела в приводимых примерах встретились только две команды, использующих стек, - это команды PUSH в начале сегмента кодов, которые обеспечивают возврат в DOS, когда EXE-программа завершается. Естественно для этих программ требуется стек очень малого размера. Однако,

команда CALL автоматически записывает в стек относительный адрес команды, следующей непосредственно за командой CALL, и увеличивает после этого указатель вершины стека. В вызываемой процедуре команда RET использует этот адрес для возврата в вызывающую процедуру и при этом автоматически уменьшается указатель вершины стека.

Таким образом, команды PUSH записывают в стек двухбайтовые адреса или другие значения. Команды POP обычно выбирают из стека записанные в него слова. Эти операции изменяют относительный адрес в регистре SP (т.е. в указатели стека) для доступа к следующему слову. Данное свойство стека требует чтобы команды RET и CALL соответствовали друг другу. Кроме того, вызванная процедура может вызвать с помощью команды CALL другую процедуру, а та в свою очередь - следующую. Стек должен иметь достаточные размеры для того, чтобы хранить все записываемые в него адреса. Для большинства примеров в данной книге стек объемом в 32 слова является достаточным.

Команды PUSH, PUSHF, CALL, INT, и INTO заносят в стек адрес возврата или содержимое флагового регистра. Команды POP, POPF, RET и IRET извлекают эти адреса или флаги из стека.

При передаче управления в EXE-программу система устанавливает в регистрах следующие значения:

DS и ES: Адрес префикса программного сегмента - область в 256 (шест.100) байт, которая предшествует выполняемому программному модулю в памяти.

CS: Адрес точки входа в программу (адрес первой выполняемой команды).

IP: Нуль.

SS: Адрес сегмента стека.

SP: Относительный адрес, указывающий на вершину стека. Например, для стека в 32 слова (64 байта), определенного как

```
            DW   32 DUP(?)
```

SP содержит 64, или шест.40.

Выполним трассировку простой EXE-программы, приведенной на рис.7.4. На практике вызываемые процедуры содержат любое число команд.

```
                TITLE   CALLPROC (EXE) Вызов процедур
0000              STACKSG SEGMENT PARA STACK 'Stack'
0000    20 [ ???? ]         DW      32 DUP(?)
0040              STACKG  ENDS

0000              CODESG  SEGMENT PARA 'Code'
0000              BEGIN   PROC    FAR
                      ASSUME  CS:CODESG,SS:STACKSG
0000  1E                  PUSH    DS
0001  2B C0                SUB     AX,AX
0003  50                  PUSH    AX
0004  E8 0008 R               CALL    B10     ;Вызвать B10
                ;       ...
0007  CB                  RET            ;Завершить программу
0008              BEGIN   ENDP
                      ;-----------------------------------
0008              B10     PROC
0008  E8 000C R               CALL    C10     ;Вызвать C10
```

```
                ;       ...
000B  C3                 RET           ;Вернуться в
000C              B10     ENDP          ; вызывающую программу
                ;------------------------------------------
000C              C10     PROC
                ;       ...
000C  C3                 RET           ;Вернуться в
000D              C10     ENDP          ; вызывающую программу
                ;------------------------------------------
000D              CODESG  ENDS
                          END     BEGIN
```

Рис.7.4. Воздействие выполнения программы на стек.

Текущая доступная ячейка стека для занесения или извлечения слова является вершина стека. Первая команда PUSH уменьшает значение SP на 2 и заносит содержимое регистра DS (в данном примере 049f) в вершину стека, т.е. по адресу 4B00+3E. Вторая команда PUSH также уменьшает значение SP на 2 и записывает содержимое регистра AX (0000) по адресу 4B00+3C. Команда CALL B10 уменьшает значение SP и записывает относительный адрес следующей команды (0007) в стек по адресу 4B00+3A. Команда CALL C10 уменьшает значение SP и записывает относительный адрес следующей команды (000B) в стек по адресу 4B00+38.

При возврате из процедуры C10 команда RET извлекает 000B из стека (4B00+38), помещает его в указатель команд IP и увеличивает значение SP на 2. При этом происходит автоматический возврат по относительному адресу 000B в кодовом сегменте, т.е. в процедуру B10.

Команда RET в конце процедуры B10 извлекает адрес 0007 из стека (4B00+3A), помещают его в IP и увеличивает значение SP на 2. При этом происходит автоматический возврат по относительному адресу 0007 в кодовом сегменте. Команда RET по адресу 0007 завершает выполнение программы, осуществляя возврат типа FAR.

Ниже показано воздействие на стек при выполнении каждой команды. Для трассировки программы можно использовать отладчик DEBUG. Приведено только содержимое памяти с адреса 0034 до 003F и содержимое регистра SP:

```
      Команда                  Стек              SP

    Начальное значение:     xxxx xxxx xxxx xxxx xxxx xxxx   0040
    PUSH DS  (запись 049F)  xxxx xxxx xxxx xxxx xxxx 049F   003E
    PUSH AX  (запись 0000)  xxxx xxxx xxxx xxxx 0000 049F   003C
    CALL B10 (запись 0007)  xxxx xxxx xxxx 0700 0000 049F   003A
    CALL C10 (запись 000B)  xxxx xxxx 0B00 0700 0000 049F   0038
    RET      (выборка 000B) xxxx xxxx xxxx 0700 0000 049F   003A
    RET      (выборка 0007) xxxx xxxx xxxx xxxx 0000 049F   003C
                            |    |    |    |    |    |
    Смещение в стеке:       0034 0036 0038 003A 003C 003E
```

Обратите внимание на два момента. Во-первых, слова в памяти содержат байты в обратной последовательности, так 0007 записывается в виде 0700. Во-вторых, отладчик DEBUG при использовании его для просмотра стека заносит в стек другие значения, включая содержимое IP, для собственных нужд.

ПРОГРАММА: РАСШИРЕННЫЕ ОПЕРАЦИИ ПЕРЕСЫЛКИ

В предыдущих программах были показаны команды пересылки

непосредственных данных в регистр, пересылки данных из памяти в регистр, пересылки содержимого регистра в память и пересылки содержимого одного регистра в другой. Во всех случаях длина данных была ограничена одним или двумя байтами и не предусмотрена пересылка данных из одной области памяти непосредственно другую область. В данном разделе объясняется процесс пересылки данных, которые имеют длину более двух байт. В гл.11 будет показано использование операций над строками для пересылки данных из одной области памяти непосредственно в другую область.

В EXE-программе, приведенной на рис.7.5, сегмент данных содержит три девятибайтовых поля, NAME1, NAME2, NAME3. Цель программы - переслать данные из поля NAME1 в поле NAME2 и переслать данные из поля NAME2 в поле NAME3. Так как эти поля имеют длину девять байт каждая, то для пересылки данных кроме простой команды MOV потребуются еще другие команды. Программа содержит несколько новых особенностей.

```
       page     65,132
TITLE   EXMOVE   (EXE) Операции расширенной пересылки
;----------------------------------------------------
STACKSG SEGMENT  PARA STACK 'Stack'
       DW       32 DUP(?)
STACKSG ENDS
;----------------------------------------------------
DATASG  SEGMENT  PARA 'Data'
NAME1   DB       'ABCDEFGHI'
NAME2   DB       'JKLMNOPQR'
NAME3   DB       'STUVWXYZ*'
DATASG  ENDS
;----------------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
       ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASG
       PUSH    DS
       SUB     AX,AX
       PUSH    AX
       MOV     AX,DATASG
       MOV     DS,AX
       MOV     ES,AX
       CALL    B10MOVE        ;Вызвать JUMP подпрограмму
       CALL    C10MOVE        ;Вызвать CALL подпрограмму
       RET                    ;Завершить программу
BEGIN   ENDP

;       Расширенная пересылка (JUMP-подпрограмма),
;       использующая переход по условию:
;       ---------------------------------------
B10MOVE PROC
       LEA     SI,NAME1       ;Инициализация адресов
       LEA     DI,NAME2       ; NAME1 и NAME2
       MOV     CX,09          ;Переслать 9 символов
B20:
       MOV     AL,[SI]        ;Переслать из NAME1
       MOV     [DI],AL        ;Переслать в NAME2
       INC     SI             ;Следующий символ в NAME1
       INC     DI             ;Следующая позиция в NAME2
       DEC     CX             ;Уменьшить счетчик цикла
       JNZ     B20            ;Счетчик > 0? Да - цикл
       RET                    ;Если счетчик = 0, то
B10MOVE ENDP                  ; вернуться
```

```
;       Расширенная пересылка (LOOP-подпрограмма),
;       использующая команду LOOP:
;       ----------------------------------------;

C10MOVE PROC
        LEA     SI,NAME2        ;Инициализация адресов
        LEA     DI,NAME3        ; NAME2 и NAME3
        MOV     CX,09           ;Переслать 9 символов
C20
        MOV     AL,[SI]         ;Переслать из NAME2
        MOV     [DI],AL         ;Переслать в NAME3
        INC     DI              ;Следующий символ в NAME2
        INC     SI              ;Следующая позиция в NAME3
        LOOP    C20             ;Уменьшить счетчик,
                                ; если не ноль, то цикл
        RET                     ;Если счетчик = 0, то
C10MOVE ENDP                    ; вернуться
CODESG  ENDS
        END     BEGIN
```

Рис.7.5. Расширенные операции пересылки.

Процедура BEGIN инициализирует сегментные регистры и затем вызывает процедуры B10MOVE и C10MOVE. Процедура B10MOVE пересылает содержимое поля NAME1 в поле NAME2. Так как каждый раз пересылается только один байт, то процедура начинает с самого левого байта в поле NAME1 и в цикле пересылает затем второй байт, третий и т.д.:

```
NAME1:   A  B  C  D  E  F  G  H  I
         |  |  |  |  |  |  |  |  |
NAME2:   J  K  L  M  N  O  P  Q  R
```

Для продвижения в полях NAME1 и NAME2 в регистр CX заносится значение 9, а регистры SI и DI используются в качестве индексных. Две команды LEA загружают относительные адреса полей NAME1 и NAME2 в регистры SI и DI:

```
LEA  SI,NAME1  ;Загрузка относительных адресов
LEA  DI,NAME2  ;  NAME1 и NAME2
```

Для пересылки содержимого первого байта из поля NAME1 в первый байт поля NAME2 используются адреса в регистрах SI и DI. kвадратные скобки в командах MOV обозначают, что для доступа к памяти используется адрес в регистре, указанном в квадратных скобках. Таким образом, команда

```
MOV  AL,[SI]
```

означает: использовать адрес в регистре SI (т.е.NAME1) для пересылки соответствующего байта в регистр AL. А команда

```
MOV  [DI],AL
```

означает: пересылать содержимое регистра AL по адресу, лежащему в регистре DI (т.е. NAME2).

Следующие команды увеличивают значения регистров SI и DI и уменьшают значение в регистре SH. Если в регистре CX не нулевое значение, управление передается на следующий цикл (на метку B20). Так как содержимое регистров SI и DI было увеличено на 1, то следующие команды MOV будут иметь дело с адресами NAME1+1 и NAME2+1. Цикл продолжается таким образом, пока не будет передано содержимое NAME1+8 и NAME2+8.

Процедура C10MOVE аналогична процедуре B10MOVE с двумя исключениями: она пересылает данные из поля NAME2 в поле NAME3 и использует команду LOOP вместо DEC и JNZ.

Задание: Введите программу, приведенную на рис.7.5, выполните ее ассемблирование, компоновку и трассировку с помощью отладчика DEBUG. Обратите внимание на изменения в регистрах, командном указателе и в стеке. Для просмотра изменений в полях NAME2 и NAME3 используйте команду D DS:0.

КОМАНДЫ ЛОГИЧЕСКИХ ОПЕРАЦИЙ: AND, OR, XOR, TEST, NOT

Логические операции являются важным элементом в проектировании микросхем и имеют много общего в логике программирования. Команды AND, OR, XOR и TEST - являются командами логических операций. Эти команды используются для сброса и установки бит и для арифметических операций в коде ASCII (см.гл.13). Все эти команды обрабатывают один байт или одно слово в регистре или в памяти, и устанавливают флаги CF, OF, PF, SF, ZF.

AND: Если оба из сравниваемых битов равны 1, то результат равен 1; во всех остальных случаях результат - 0.

OR: Если хотя бы один из сравниваемых битов равен 1, то результат равен 1; если сравниваемые биты равны 0, то результат - 0.

XOR: Если один из сравниваемых битов равен 0, а другой равен 1, то результат равен 1; если сравниваемые биты одинаковы (оба - 0 или оба - 1) то результат - 0.

TEST: действует как AND-устанавливает флаги, но не изменяет биты.

Первый операнд в логических командах указывает на один байт или слово в регистре или в памяти и является единственным значением, которое может изменятся после выполнения команд. В следующих командах AND, OR и XOR используются одинаковые битовые значения:

```
                     AND      OR       XOR
                     0101     0101     0101
                     0011     0011     0011
                     ----     ----     ----
         Результат:  0001     0111     0110
```

Для следующих несвязанных примеров, предположим, что AL содержит 1100 0101, а BH содержит 0101 1100:

```
1.       AND  AL,BH    ;Устанавливает в AL 0100 0100
2.       OR   BH,AL    ;Устанавливает в BH 1101 1101
3.       XOR  AL,AL    ;Устанавливает в AL 0000 0000
4.       AND  AL,00    ;Устанавливает в AL 0000 0000
5.       AND  AL,0FH   ;Устанавливает в AL 0000 0101
6.       OR   CL,CL    ;Устанавливает флаги SF и ZF
```

Примеры 3 и 4 демонстрируют способ очистки регистра. В примере 5 обнуляются левые четыре бита регистра AL. Хотя команды сравнения CMP могут быть понятнее, можно применить команду OR для следующих целей:

```
1.       OR   CX,CX    ;Проверка CX на нуль
         JZ   ...      ;Переход, если нуль
2.       OR   CX,CX    ;Проверка знака в CX
         JS   ...      ;Переход, если отрицательно
```

Команда TEST действует аналогично команде AND, но устанавливает только флаги, а операнд не изменяется. Ниже приведено несколько примеров:

```
1.      TEST BL,11110000B   ;Любой из левых бит в BL
        JNZ  ...            ; равен единице?
2.      TEST AL,00000001B   ;Регистр AL содержит
        JNZ  ...            ; нечетное значение?
3.      TEST DX,OFFH        ;Регистр DX содержит
        JZ   ...            ; нулевое значение?
```

Еще одна логическая команда NOT устанавливает обратное значение бит в байте или в слове, в регистре или в памяти: нули становятся единицами, а единицы - нулями. Если, например, регистр AL содержит 1100 0101, то команда NOT AL изменяет это значение на 0011 1010. Флаги не меняются. Команда NOT не эквивалентна команде NEG, которая меняет значение с положительного на отрицательное и наоборот, посредством замены бит на противоположное значение и прибавления единицы (см. "Отрицательные числа" в гл.1.).

ПРОГРАММА: ИЗМЕНЕНИЕ СТРОЧНЫХ БУКВ НА ЗАГЛАВНЫЕ

Существуют различные причины для преобразований между строчными и заглавными буквами. Например, вы могли получить файл данных, созданный на компьютере, который работает только с заглавными буквами. Или некая программа должна позволить пользователям вводить команды как заглавными, так и строчными буквами (например, YES или yes) и преобразовать их в заглавные для проверки. Заглавные буквы от A до Z имеют шест. коды от 41 до 5A, а строчные буквы от a до z имеют шест. коды от 61 до 7A. Единственная разница в том, что пятый бит равен 0 для заглавных букв и 1 для строчных:

```
     Биты:  76543210       Биты:  76543210
  Буква A:  01000001    Буква a:  01100001
  Буква Z:  01011010    Буква z:  01111010
```

COM-программа, приведенная на рис.7.6, преобразует данные в поле TITLEX из строчных букв в прописные, начиная с адреса TITLEX+1. Программа инициализирует регистр BX адресом TITLEX+1 и использует его для пересылки символов в регистр AH, начиная с TITLEX+1. Если полученное значение лежит в пределах от шест.61 и до 7A, то команда AND устанавливает бит 5 в 0:

```
        AND  AH,11011111B
```

Все символы, отличные от строчных букв (от a до z), не изменяются. Измененные символы засылаются обратно в область TITLEX, значение в

регистре BX увеличивается для очередного символа и осуществляется переход на следующий цикл.

```
                    TITLE   CASE    (COM) Перекод. в заглавные буквы
0000                  CODESG  SEGMENT PARA 'CODE'
                          ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
0001                        ORG     100H
0001  EB 1C 90          BEGIN:  JMP     MAIN
                    ; ------------------------------------------
0003  43 68 61 6E 67 65  TITLEX  DB      'Change to uppercase letters'
      20 74 6F 20 75 70
```

```
    70 65 72 63 61 73
    65 20 6C 65 74 74
    65 72 73
                    ; -----------------------------------------
011E               MAIN    PROC    NEAR
011E  8D 1E 0104 R            LEA     BX,TITLEX+1  ;Адрес первого симв.
0122  B9 001F               MOV     CX,31        ;Число символов
0125               B20:
0125  8A 27                MOV     AH,[BX]      ;Символ из TITLEX
0127  80 FC 61             CMP     AH,61H       ;Это
012A  72 0A                JB      B30          ; прописная
012C  80 FC 7A             CMP     AH,7AH       ; буква
012F  77 05                JA      B30          ; ?
0131  80 E4 DF             AND     AH,11011111B ;Да - преобразовать
0134  88 27                MOV     [BX],AH      ;Записать в TITLEX
0136               B30:
0136  43                   INC     BX           ;Следующий символ
0137  E2 EC                LOOP    B20          ;Повторить цикл 31 раз
0139  C3                   RET
013A               MAIN    ENDP
013A               CODESG  ENDS
                       END     BEGIN
```

Рис.7.6. Изменение строчных букв на прописные.

Используемый таким образом регистр BX действует как индексный регистр для адресации в памяти. Для этих целей можно использовать также регистры SI и DI.

КОМАНДЫ СДВИГА И ЦИКЛИЧЕСКОГО СДВИГА

Команды сдвига и циклического сдвига, которые представляют собой часть логических возможностей компьютера, имеют следующие свойства:

- обрабатывают байт или слово;
- имеют доступ к регистру или к памяти;
- сдвигают влево или вправо;
- сдвигают на величину до 8 бит (для байта) и 16 бит (для слова)
- сдвигают логически (без знака) или арифметически (со знаком).

Значение сдвига на 1 может быть закодировано как непосредственный операнд, значение больше 1 должно находиться в регистре CL.

Команды сдвига

При выполнении команд сдвига флаг CF всегда содержит значение последнего выдвинутого бита. Существуют следующие команды сдвига:

```
SHR  ;Логический (беззнаковый) сдвиг вправо
SHL  ;Логический (беззнаковый) сдвиг влево
SAR  ;Арифметический сдвиг вправо
SAL  ;Арифметический сдвиг влево
```

Следующий фрагмент иллюстрирует выполнение команды SHR:

```
MOV  CL,03         ;    AX:
```

```
MOV  AX,10110111B   ; 10110111
SHR  AX,1           ; 01011011   ;Сдвиг вправо на 1
SHR  AX,CL          ; 00001011   ;Сдвиг вправо на 3
```

Первая команда SHR сдвигает содержимое регистра AX вправо на 1 бит. Выдвинутый в результате один бит попадает в флаг CF, а самый левый бит регистра AX заполняется нулем. Вторая команда сдвигает содержимое регистра AX еще на три бита. При этом флаг CF последовательно принимает значения 1, 1, 0, а в три левых бита в регистре AX заносятся нули.

Рассмотрим действие команд арифметического вправо SAR:

```
MOV  CL,03          ;    AX:
MOV  AX,10110111B   ; 10110111
SAR  AX,1           ; 11011011   ;Сдвиг вправо на 1
SAR  AX,CL          ; 11111011   ;Сдвиг вправо на 3
```

Команда SAR имеет важное отличие от команды SHR: для заполнения левого бита используется знаковый бит. Таким образом, положительные и отрицательные величины сохраняют свой знак. В приведенном примере знаковый бит содержит единицу.

При сдвигах влево правые биты заполняются нулями. Таким образом, результат команд сдвига SHL и SAL индентичен.

Сдвиг влево часто используется для удваивания чисел, а сдвиг вправо - для деления на 2. Эти операции осуществляются значительно быстрее, чем команды умножения или деления. Деление пополам нечетных чисел (например, 5 или 7) образует меньшие значения (2 или 3, соответственно) и устанавливает флаг CF в 1. Кроме того, если необходимо выполнить сдвиг на 2 бита, то использование двух команд сдвига более эффективно, чем использование одной команды с загрузкой регистра CL значением 2.

Для проверки бита, занесенного в флаг CF используется команда JC (переход, если есть перенос).

Команды циклического сдвига

Циклический сдвиг представляет собой операцию сдвига, при которой выдвинутый бит занимает освободившийся разряд. Существуют следующие команды циклического сдвига:

```
ROR  ;Циклический сдвиг вправо
ROL  ;Циклический сдвиг влево
RCR  ;Циклический сдвиг вправо с переносом
RCL  ;Циклический сдвиг влево с переносом
```

Следующая последовательность команд иллюстрирует операцию циклического сдвига ROR:

```
MOV  CL,03          ;    BX:
MOV  BX,10110111B   ; 10110111
ROR  BX,1           ; 11011011   ;Сдвиг вправо на 1
ROR  BX,CL          ; 01111011   ;Сдвиг вправо на 3
```

Первая команда ROR при выполнении циклического сдвига переносит правый единичный бит регистра BX в освободившуюся левую позицию. Вторая команда ROR переносит таким образом три правых бита.

В командах RCR и RCL в сдвиге участвует флаг CF. Выдвигаемый из регистра бит заносится в флаг CF, а значение CF при этом поступает в освободившуюся позицию.

Рассмотрим пример, в котором используются команды циклического и простого сдвига. Предположим, что 32-битовое значение находится в регистрах DX:AX так, что левые 16 бит лежат в регистре DX, а правые - в

AX. Для умножения на 2 этого значения возможны следующие две команды:

```
        SHL  AX,1      ;Умножение пары регистров
        RCL  DX,1      ;  DX:AX на 2
```

Здесь команда SHL сдвигает все биты регистра AX влево, причем самый левый бит попадает в флаг CF. Затем команда RCL сдвигает все биты регистра DX влево и в освободившийся правый бит заносит значение из флага CF.

ОРГАНИЗАЦИЯ ПРОГРАММ

Ниже даны основные рекомендации для написания ассемблерных программ:

1. Четко представляйте себе задачу, которую должна решить программа.
2. Сделайте эскиз задачи в общих чертах и спланируйте общую логику программы. Например, если необходимо проверить операции пересылки нескольких байт (как в примере на рис.7.5), начните с определения полей с пересылаемыми данными. Затем спланируйте общую стратегию для инициализации, условного перехода и команды LOOP. Приведем основную логику, которую используют многие программисты в таком случае:

```
    инициализация стека и сегментных регистров
    вызов подпрограммы цикла
    возврат
```

Подпрограмма цикла может быть спланирована следующим образом:

```
    инициализация регистров значениями адресов
    и числа циклов

Метка:    пересылка одного байта
        увеличение адресов на 1
        уменьшение счетчика на 1:
            если счетчик не ноль, то идти на метку
            если ноль, возврат
```

3. Представьте программу в виде логических блоков, следующих друг за другом. Процедуры не превышающие 25 строк (размер экрана) удобнее для отладки.
4. Пользуйтесь тестовыми примерами программ. Попытки запомнить все технические детали и программирование сложных программ "из головы" часто приводят к многочисленным ошибкам.
5. Используйте комментарии для описания того, что должна делать процедура, какие арифметические действия или операции сравнения будут выполняться и что делают редко используемые команды. (Например, команда XLAT, не имеющая операндов).
6. Для кодирования программы используйте заготовку программы, скопированной в файл с новым именем.

В следующих программах данной книги важным является использование команды LEA, индексных регистров SI и DI, вызываемых процедур. Получив теперь базовые знания по ассемблеру, можем перейти к более развитому и полезному программированию.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Метки процедур (например, B20:) должны завершаться двоеточием для указания типа NEAR. Отсутствие двоеточия приводит к ассемблерной ошибке.

- Метки для команд условного перехода и LOOP должны лежать в границах -128 до +127 байт. Операнд таких команд генерирует один байт объектного кода. Шест. от 01 до 7F соответствует десятичным значениям от +1 до +127, а шест. от FF до 80 покрывает значения от -1 до +128. Так как длина машинной команды может быть от 1 до 4 байт, то соблюдать границы не просто. Практически можно ориентироваться на размер в два экрана исходного текста (примерно 50 строк).

- При использовании команды LOOP, инициализируйте регистр CX положительным числом. Команда LOOP контролирует только нулевое значение, при отрицательном программа будет продолжать циклиться.

- Если некоторая команда устанавливает флаг, то данный флаг сохраняет это значение, пока другая команда его не изменит. Например, если за арифметической командой, которая устанавливает флаги, следуют команды MOV, то они не изменят флаги. Однако, для минимизации числа возможных ошибок, следует кодировать команды условного перехода непосредственно после команд, устанавливающих проверяемые флаги.

- Выбирайте команды условного перехода соответственно операциям над знаковыми или беззнаковыми данными.

- Для вызова процедуры используйте команду CALL, а для возврата из процедуры - команду RET. Вызываемая процедура может, в свою очередь, вызвать другую процедуру, и если следовать существующим соглашениям, то команда RET всегда будет выбирать из стека правильный адрес возврата. Единственные примеры в этой книге, где используется переход в процедуру вместо ее вызова - в начале COM-программ.

- Будьте внимательны при использовании индексных операндов. Сравните:

```
MOV  AX,SI
MOV  AX,[SI]
```

Первая команда MOV пересылает в регистр AX содержимое регистра SI. Вторая команда MOV для доступа к пересылаемому слову в памяти использует относительный адрес в регистре SI.

- Используйте команды сдвига для удваивания значений и для деления пополам, но при этом внимательно выбирайте соответствующие команды для знаковых и беззнаковых данных.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

7.1. Какое максимальное количество байт могут обойти команды короткий JMP, LOOP и относительный переход? Какой машинный код операнда при этом генерируется?

7.2. Команда JMP начинается на шест.0624. Определите адрес перехода, если шест. объектный код для операнда команды JMP: а) 27, б) 6B, в) C6.

7.3. Напишите программу вычисления 12 чисел Фибоначчи: 1, 1, 2, 3, 5, 8, 13,... (каждое число в последовательности представляет собой сумму двух предыдущих чисел). Для организации цикла используйте команду LOOP. Выполните ассемблирование, компоновку и с помощью отладчика DEBUG

трассировку программы.

7.4. Предположим, что регистры AX и BX содержат знаковые данные, а CX и DX - беззнаковые. Определите команды CMP (где необходимо) и команды безусловного перехода для следующих проверок:

а) значение в DX больше, чем в CX?
б) значение в BX больше, чем в AX?
в) CX содержит нуль?
г) было ли переполнение?
д) значение в BX равно или меньше, чем в AX?
е) значение в DX равно или меньше, чем в CX?

7.5. На какие флаги воздействуют следующие события и какое значение этих флагов?

а) произошло переполнение;
б) результат отрицательный;
в) результат нулевой;
г) обработка в одношаговом режиме;
д) передача данных должна быть справа налево.

7.6. Что произойдет при выполнении программы , приведенной на рис.7.4, если в процедуре BEGIN будет отсутствовать команда RET?

7.7. Какая разница между кодированием в директиве PROC операнда с типом FAR и с типом NEAR?

7.8. Каким образом может программа начать выполнение процедуры?

7.9. В EXE-программе процедура A10 вызывает B10, B10 вызывает C10, а C10 вызывает D10. Сколько адресов, кроме начальных адресов возврата в DOS, содержит стек?

7.10. Предположим , что регистр BL содержит 11100011 и поле по имени BOONO содержит 01111001. Определите воздействие на регистр BL для следующих команд: а) XOR BL,BOONO; б) AND BL,BOONO; в) OR BL,BOONO; г) XOR BL,11111111B; д) AND BL,00000000B.

7.11. Измените программу на рис.7.6 для: а) определения содержимого TITLEX заглавными буквами; б) преобразование заглавных букв в строчные.

7.12. Предположим, что регистр DX содержит 10111001 10111001, а регистр CL - 03. Определите содержимое регистра DX после следующих несвязанных команд: а) SHR DX,1; б) SHR DX,CL; в) SHL DX,CL; г) SHL DL,1; д) ROR DX,CL; е) ROR DL,CL; ж) SAL DH,1.

7.13. Используя команды сдвига, пересылки и сложения, умножьте содержимое регистра AX на 10.

7.14. Пример программы, приведенной в конце раздела "сдвиг и циклический сдвиг", умножает содержимое пары регистров DX:AX на 2. Измените программу для: а) умножения на 4; б) деления на 4; в) умножения 48 бит в регистрах DX:AX:BX на 2.

ГЛАВА 8 Экранные операции I: Основные свойства

Ц е л ь: Объяснить требования для вывода информации на экран, а также

для ввода данных с клавиатуры.

ВВЕДЕНИЕ

В предыдущих главах мы имели дело с программами, в которых данные определялись в операндах команд (непосредственные данные) или инициализировались в конкретных полях программы. Число практических применений таких программ в действительности мало. Большинство программ требуют ввода данных с клавиатуры, диска или модема и обеспечивают вывод данных в удобном формате на экран, принтер или диск. Данные, предназначенные для вывода на экран и ввода с клавиатуры, имеют ASCII формат.

Для выполнения ввода и вывода используется команда INT (прерывание). Существуют различные требования для указания системе какое действие (ввод или вывод) и на каком устройстве необходимо выполнить. Данная глава раскрывает основные требования для вывода информации на экран и ввода данных с клавиатуры.

Все необходимые экранные и клавиатурные операции можно выполнить используя команду INT 10H, которая передает управление непосредственно в BIOS. Для выполнения некоторых более сложных операций существует прерывание более высокого уровня INT 21H, которое сначала передает управление в DOS. Например, при вводе с клавиатуры может потребоваться подсчет введенных символов, проверку на максимальное число символов и проверку на символ Return. Прерывание DOS INT 21H выполняет многие из этих дополнительных вычислений и затем автоматически передает управление в BIOS.

Материал данной главы подходит как для монохромных (черно-белых, BW), так и для цветных видеомониторов. В гл. 9 и 10 приведен материал для управления более совершенными экранами и для использования цвета.

КОМАНДА ПРЕРЫВАНИЯ: INT

Команда INT прерывает обработку программы, передает управление в DOS или BIOS для определенного действия и затем возвращает управление в прерванную программу для продолжения обработки. Наиболее часто прерывание используется для выполнения операций ввода или вывода. Для выхода из программы на обработку прерывания и для последующего возврата команда INT выполняет следующие действия:

- уменьшает указатель стека на 2 и заносит в вершину стека содержимое флагового регистра;
- очищает флаги TF и IF;
- уменьшает указатель стека на 2 и заносит содержимое регистра CS в стек;
- уменьшает указатель стека на 2 и заносит в стек значение командного указателя;
- обеспечивает выполнение необходимых действий;
- восстанавливает из стека значение регистра и возвращает управление в прерванную программу на команду, следующую после INT.

Этот процесс выполняется полностью автоматически. Необходимо лишь определить сегмент стека достаточно большим для записи в него значений регистров.

В данной главе рассмотрим два типа прерываний: команду BIOS INT 10H и команду DOS INT 21H для вывода на экран и ввода с клавиатуры. В последующих примерах в зависимости от требований используются как INT 10H так и INT 21H.

УСТАНОВКА КУРСОРА

Экран можно представить в виде двумерного пространства с адресуемыми позициями в любую из которых может быть установлен курсор. Обычный видеомонитор, например, имеет 25 строк (нумеруемых от 0 до 24) и 80 столбцов (нумеруемых от 0 до 79). В следующей таблице приведены некоторые примеры положений курсора на экране:

Положение	Дес. формат		Шест. формат	
	строка	столбец	строка	столбец
Верхний левый угол	00	00	00	00
Верхний правый угол	00	79	00	4F
Центр экрана	12	39/40	00	27/28
Нижний левый угол	24	00	18	00
Нижний правый угол	24	79	18	4F

Команда INT 10H включает в себя установку курсора в любую позицию и очистку экрана. Ниже приведен пример установки курсора на 5-ую строку и 12-ый столбец:

```
MOV  AH,02     ;Запрос на установку курсора
MOV  BH,00     ;Экран 0
MOV  DH,05     ;Строка 05
MOV  DL,12     ;Столбец 12
INT  10H       ;Передача управления в BIOS
```

Значение 02 в регистре AH указывает команде INT 10H на выполнение операции установки курсора. Значение строки и столбца должны быть в регистре DX, а номер экрана (или страницы) в регистре BH (обычно 0). Содержимое других регистров несущественно. Для установки строки и столбца можно также использовать одну команду MOV с непосредственным шест. значением:

```
MOV  DX,050CH  ;Строка 5, столбец 12
```

ОЧИСТКА ЭКРАНА

Запросы и команды остаются на экране пока не будут смещены в результате прокручивания ("скроллинга") или переписаны на этом же месте другими запросами или командами. Когда программа начинает свое выполнение, экран может быть очищен. Очищаемая область экрана может начинаться в любой позиции и заканчиваться в любой другой позиции с большим номером. Начальное значение строки и столбца заносится в регистр DX, значение 07 - в регистр BH и 0600H в AX. В следующем примере выполняется очистка всего экрана:

```
MOV  AX,0600H  ;AH 06 (прокрутка)
               ;AL 00 (весь экран)
MOV  BH,07     ;Нормальный атрибут (черно/белый)
MOV  CX,0000   ;Верхняя левая позиция
MOV  DX,184FH  ;Нижняя правая позиция
INT  10H       ;Передача управления в BIOS
```

Значение 06 в регистре AH указывает команде INT 10H на выполнение операции очистки экрана. Эта операция очищает экран пробелами; в следующей главе скроллинг (прокрутка) будет рассмотрен подробнее. Если вы по ошибке установили нижнюю правую позицию больше, чем шест. 184F, то очистка перейдет вновь к началу экрана и вторично заполнит некоторые позиции пробелами. Для монохромных экранов это не вызывает каких-либо неприятностей, но для некоторых цветных мониторов могут возникнуть серьезные ошибки.

ЭКРАННЫЕ И КЛАВИАТУРНЫЕ ОПЕРАЦИИ: БАЗОВАЯ ВЕРСИЯ DOS

Обычно программы должны выдать на экран сообщение о завершении или об обнаружении ошибки, отобразить запрос для ввода данных или для получения указания пользователя. Рассмотрим сначала методы, применяемые в базовой версии DOS, в последующих разделах будут показаны расширенные методы, введенные в DOS версии 2.0. Операции из базовой DOS работают во всех версиях, хотя в руководстве по DOS рекомендуется применять расширенные возможности для новых разработок. В базовой версии DOS команды вывода на экран более сложны, но команды ввода с клавиатуры проще в использовании, благодаря встроенным проверкам.

ВЫВОД НА ЭКРАН: БАЗОВАЯ ВЕРСИЯ DOS

Вывод на экран в базовой версии DOS требует определения текстового сообщения в области данных, установки в регистре AH значения 09 (вызов функции DOS) и указания команды DOS INT 21H. В процессе выполнения операции конец сообщения определяется по ограничителю ($), как это показано ниже:

```
        NAMPRMP     DB   'Имя покупателя?','$'
                    .
                    .
                    MOV  AH,09          ;Запрос вывода на экран
                    LEA  DX,NAMPRMP     ;Загрузка адреса сообщ.
                    INT  21H            ;Вызов DOS
```

Знак ограничителя "$" можно кодировать непосредственно после символьной строки (как показано в примере), внутри строки: 'Имя покупателя?$', или в следующем операторе DB '$'. Используя данную операцию, нельзя вывести на экран символ доллара "$". Кроме того, если знак доллара будет отсутствовать в конце строки, то на экран будут выводиться все последующие символы, пока знак "$" не встретиться в памяти.

Команда LEA загружает адрес области NAMPRMP в регистр DX для передачи в DOS адреса выводимой информации. Адрес поля NAMPRMP, загружаемый в DX по команде LEA, является относительным, поэтому для вычисления абсолютного адреса данных DOS складывает значения регистров DS и DX (DS:DX).

ПРОГРАММА: ВЫВОД НА ЭКРАН НАБОРА СИМВОЛОВ КОДА ASCII

Большинство из 256 кодов ASCII имеют символьное представление, и могут быть выведены на экран. Шест. коды 00 и FF не имеют символов и выводятся на экран в виде пробелов, хотя символ пробела имеет в ASCII шест. код 20.

На рис.8.1 показана COM-программа, которая выводит на экран полный набор символов кода ASCII. Программа вызывает три процедуры; B10CLR,

C10SET и D10DISP. Процедура B10CLR очищает экран, а процедура C10SET устанавливает курсор в положение 00,00. Процедура D10DISP выводит содержимое поля CTR, которое в начале инициализировано значением 00 и затем увеличивается на 1 при каждом выводе на экран, пока не достигнет шест. значения FF.

```
       page    60,132
TITLE   ALLASC  (COM) Вывод на экран ASCII-символов 00-FF
CODESC  SEGMENT PARA 'Code'
       ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:NOTHING
       ORG     100H
BEGIN:  JMP     SHORT MAIN
CTR     DB      00,'S'

;              Основная процедура:
;              ------------------
MAIN    PROC    NEAR
       CALL    B10CDR          ;Очистить экран
       CALL    C10SET          ;Установить курсор
       CALL    D10DISP         ;Вывести символ на экран
       RET
MAIN    ENDP
;              Очистка экрана:
;              --------------
B10CLR  PROC
       MOV     AX,0600H
       MOV     BH,07
       MOV     CX,0000         ;Левая верхняя позиция
       MOV     DX,184FH        ;Правая нижняя позиция
       INT     10H
       RET
B10CLR  ENDP
;              Установка курсора в 00,00:
;              -------------------------
C10SET  PROC
       MOV     AN,02
       MOV     BN,00
       MOV     DX,0000
       INT     10H
       RET
C10SET  ENDP
;              Вывод на экран ASCII символов:
;              -----------------------------
D10DISP PROC
       MOV     CX,256          ;256 итераций
       LEA     DX,CTR          ;Адрес счетчика
D20
       MOV     AH,09           ;Функция вывода символа
       INT     21H
       INC     CTR             ;Увеличить счетчик
       LOOP    D20             ;Уменьшить CX,
                               ; цикл, если не ноль
       RET                     ;Вернуться
D10DISP ENDP

CODESG  ENDS
       END     BEGIN
```

Рис.8.1. Вывод на экран набора символов кода ASCII

Так как символ доллара не выводится на экран и кроме того коды от шест.08 до шест.0D являются специальными управляющими символами, то это приводит к перемещению курсора и другим управляющим воздействиям. Задание: введите программу (рис.8.1), выполните ассемблирование, компоновку и преобразование в COM-файл. Для запуска программы введите ее имя, например, B:ASCII.COM.

Первая выведенная строка начинается с пробельного символа (шест.00), двух "улыбающихся лиц" (шест.01 и 02) и трех карточных символов (шест.03, 04 и 05). Код 07 выдает звуковой сигнал. Код 06 должен отобразиться карточным символом "пики", но управляющие символы от шест.08 до 0D сотрут его. Код 0D является "возвратом каретки" и приводит к переходу на новую (следующую)строку. Код шест.0E - представляется в виде музыкальной ноты. Символы после шест.7F являются графическими.

Можно изменить программу для обхода управляющих символов. Ниже приведен пример фрагмента программы, позволяющий обойти все символы между шест.08 и 0D. Вы можете поэкспериментировать, обходя только, скажем, шест.08 (возврат на символ) и 0D (возврат каретки):

```
        CMP  CTR,08H        ;Меньше чем 08?
        JB   D30            ; да - принять
        CMP  CTR,0DH        ; Меньше/равно 0D?
        JBE  D40            ; да - обойти
D30:
        MOV  AH,40H         ;Вывод символов < 08
        ...                 ;          и > 0D
        INT  21H
D40:
        INC  CTR
```

ВВОД ДАННЫХ С КЛАВИАТУРЫ: БАЗОВАЯ ВЕРСИЯ DOS

Процедура ввода данных с клавиатуры проще, чем вывод на экран. Для ввода, использующего базовую DOS, область ввода требует наличия списка параметров, содержащего поля, которые необходимы при выполнении команды INT. Во-первых, должна быть определена максимальная длина вводимого текста. Это необходимо для предупреждения пользователя звуковым сигналом, если набран слишком длинный текст; символы, превышающие максимальную длину не принимаются. Во-вторых, в списке параметров должно быть определенное поле, куда команда возвращает действительную длину введенного текста в байтах.

Ниже приведен пример, в котором определен список параметров для области ввода. LABEL представляет собой директиву с атрибутом BYTE. Первый байт содержит максимальную длину вводимых данных. Так как это однобайтовое поле, то возможное максимальное значение его - шест.FF или 255. Второй байт необходим DOS для занесения в него действительного числа введенных символов. Третьим байтом начинается поле, которое будет содержать введенные символы.

```
NAMEPAR    LABEL    BYTE          ;Список параметров:
MAXLEN     DB       20            ; Максимальная длина
ACTLEN     DB       ?             ; Реальная длина
NAMEFLD    DB       20 DUP (' ')  ; Введенные символы
```

Так как в списке параметров директива LABEL не занимает места, то NAMEPAR и MAXLEN указывают на один и тот же адрес памяти. В трансляторе MASM для определения списка параметров в виде структуры может

использоваться также директива STRUC. Однако, в связи с тем, что ссылки на имена, определенные внутри, требуют специальной адресации, воздержимся сейчас от рассмотрения данной темы до гл.24 "Директивы ассемблера".

Для запроса на ввод необходимо поместить в регистр AH номер функции - 10 (шест. 0AH), загрузить адрес списка параметров (NAMEPAR в нашем примере) в регистр DX и выполнить INT 21H:

```
MOV  AH,0AH         ;Запрос функции ввода
LEA  DX,NAMEPAR     ;Загрузить адреса списка параметров
INT  21H            ;Вызвать DOS
```

Команда INT ожидает пока пользователь не введет с клавиатуры текст, проверяя при этом, чтобы число введенных символов не превышало максимального значения, указанного в списке параметров (20 в нашем примере). Для указания конца ввода пользователь нажимает клавишу Return. Код этой клавиши (шест. 0D) также заносится в поле ввода (NAMEFLD в нашем примере). Если, например, пользователь ввел имя BROWN (Return), то список параметров будет содержать информацию:

```
дес.:  |20| 5| B| R| O| W| N| #|  |  |  |  | ...
шест.: |14|05|42|52|4F|57|4E|0D|20|20|20|20| ...
```

Во второй байт списка параметров (ACTLEN в нашем примере) команда заносит длину введенного имени - 05. Код Return находится по адресу NAMEFLD +5. Символ # использован здесь для индикации конца данных, так как шест. 0D не имеет отображаемого символа. Поскольку максимальная длина в 20 символов включает шест.0D, то действительная длина вводимого текста может быть только 19 символов.

ПРОГРАММА: ВВОД И ВЫВОД ИМЕН

EXE-программа, приведенная на рис. 8.2, запрашивает ввод имени, затем отображает в середине экрана введенное имя и включает звуковой сигнал. Программа продолжает запрашивать и отображать имена, пока пользователь не нажмет Return в ответ на очередной запрос. Рассмотрим ситуацию, когда пользователь ввел имя TED SMITH:

1. Разделим длину 09 на 2 получим 4, и
2. Вычтем это значение из 40, получим 36

Команда SHR в процедуре E10CENT сдвигает длину 09 на один бит вправо, выполняя таким образом деление на 2. Значение бит 00001001 переходит в 00000100. Команда NEG меняет знак +4 На -4. Команда ADD прибавляет значение 40, получая в регистре DL номер начального столбца - 36. При установке курсора на строку 12 и столбец 36 имя будет выведено на экран в следующем виде:

```
Строка 12:     TED SMITH
               |   |
Столбец:       36  40
```

В процедуре E10CODE имеется команда, которая устанавливает символ звукового сигнала (07) в области ввода непосредственно после имени:

```
MOV  NAMEFLD[BX],07
```

Предшествующая команда устанавливает в регистре BX значение длины, и команда MOV затем, комбинируя длину в регистре BX и адрес поля NAMEFLD, пересылает код 07. Например, при длине имени 05 код 07 будет помещен по

адресу NAMEFLD+05 (замещая значение кода Return). Последняя команда в процедуре E10CODE устанавливает ограничитель "$" после кода 07. Таким образом, когда процедура F10CENT выводит на экран имя, то генерируется также звуковой сигнал.

```
      page    60,132
TITLE   CTRNAME (EXE) Ввод имен и вывод в центр экрана
;------------------------------------------------------
STSCKSG SEGMENT PARA STACK 'Stack'
      DW      32 DUP(?)
STACKSG ENDS
;------------------------------------------------------
DATASG  SEGMENT PARA 'Data'
NAMEPAR LABEL   BYTE           ;Имя списка параметров:
MAXNLEN DB      20             ; макс. длина имени
NAMELEN DB      ?              ; число введенных символов
NAMEFLD DB      20 DUP(' '),'$' ;имя и ограничитель для вывода на экран
PRIMPT  DB      'Name? ', '$'
DATASG  ENDS
;------------------------------------------------------
CODESG  SEGMENT PARA 'Code"
BEGIN   PROC    FAR
      ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASC
      PUSH    DS
      SUB     AX,AX
      PUCH    AX
      MOV     AX,DATASC
      MOV     DS,AX
      MOV     ES,AX
      CALL    Q10CLR          ;Очистить экран
A20LOOP:
      MOV     DX,0000         ;Установить курсор в 00,00
      CALL    Q20CURS
      CALL    B10PRMP         ;Выдать текст запроса
      CALL    D10INPT         ;Ввести имя
      CALL    Q10CLR          ;Очистить экран
      CMP     NAMELEN,00      ;Имя введено?
      JE      A30             ; нет - выйти
      CALL    E10CODE         ;Установить звуковой сигнал
                              ; и ограничитель '$'
      CALL    F10CENT         ;Центрирование и вывод
      JMP     A20LOOP
      A30:
            RET               ;Вернуться в DOS
BEGIN   ENDP
;             Вывод текста запроса:
;             --------------------
B10PRMP PROC    NEAR
      MUV     AN,09           ;Функция вывода на экран
      LEA     DX,PROMPT
      INT     21H
      RET
B10PRMP ENDP
;             Ввод имени с клавиатуры:
;             ----------------------
D10INPT PROC    NEAR
      MOV     AN,0AN          ;Функция ввода
      LEA     DX,NAMEPAR
      INT     21H
```

```
       RET
D10INPT ENDP
;            Установка сигнала и ограничителя '$':
;            ----------------------------------
E10CODE PROC   NEAR
       MOV    BN,00          ;Замена символа Return (0D)
       MOV    BL,NAMELEN      ; на зв. сигнал (07)
       MOV    NAMEFLD[BX],07
       MOV    NAMEFLD[BX+1],'$' ;Установить ограничитель
       RET
E10CODE ENDP
;            Центрирование и вывод имени на экран:
;            ----------------------------------
F10CENT PROC   NEAR
       MOV    DL,NAMELEN       ;Определение столбца:
       SHR    DL,1            ; разделить длину на 2,
       NEG    DL              ; поменять знак,
       ADD    DL,40            ; прибавить 40
       MOV    DH,12            ;Центральная строка
       CALL   Q20CURS          ;Установить курсор
       MOV    AN,09
       LEA    DX,NAMEFLD       ;Вывести имя на экран
       INT    21H
       RET
F10CENT ENDP
;            Очистить экран:
;            --------------
Q10CLR  PROC   NEAR
       MOV    AX,0600H         ;Функция прокрутки экрана
       MOV    BH,30            ;Цвет (07 для ч/б)
       MOV    CX,0000          ;От 00,00
       MOV    DX,184FH         ;До 24,79
       INT    10H             ;Вызов BIOS
       RET
Q10CLR
;            Установка курсора (строка/столбец):
;            ---------------------------------
Q20CURS PROC   NEAR            ;DX уже установлен
       MOV    AH,02            ;Функция установки курсора
       MOV    BH,00            ;Страница #0
       INT    10H             ;Вызов BIOS
       RET
Q20CURS ENDP

CODESG  ENDS
       END    BEGIN
```

Рис.8.2. Ввод и отображение имен.

Ввод единственного символа Return

При вводе имени, превышающего по длине максимальное значение, указанное в списке параметров, возникает звуковой сигнал и система ожидает ввода только символа Return. Если вообще не вводить имя, а только нажать клавишу Return, то система примет ее и установит в списке параметров нулевую длину следующим образом:

```
            Список параметров (шест.):    |14|00|0D|...
```

Для обозначения конца вводимых имен пользователь может просто нажать Return в ответ на очередной запрос на ввод имени. Программа определяет конец ввода по нулевой длине.

Замена символа Return

Вводимые значения можно использовать для самых разных целей, например: для печати сообщений, сохранения в таблице, записи на диск. При этом, возможно, появится необходимость замены символа Return (шест.0D) в области NAMEFLD на символ пробела (шест.20). Поле NAMELEN содержит действительную длину или относительный адрес кода 0D. Если, например, NAMELEN содержит длину 05, то адрес кода 0D равен NAMEFLD+5. Можно занести эту длину в регистр BX для индексной адресации в поле NAMEFLD:

```
MOV  BH,00              ;Установить в регистре BX
MOV  BL,NAMELEN          ; значение 0005
MOV  NAMEFLD[BX],20H     ;Заменить 0D на пробел
```

Третья команда MOV заносит символ пробела (шест.20) по адресу, определенному первым операндом: адрес поля NAMEFLD плюс содержимое регистра BX, т.е. NAMEFLD+5.

Очистка области ввода

Вводимые символы заменяют предыдущее содержимое области ввода и остаются там, пока другие символы не заменят их. Рассмотрим следующие три успешных ввода имен:

```
Ввод            NAMEPAR (шест.)

1. BROWN    |14|05|42|52|4F|57|4E|0D|20|20|20| ... |20|
2. HAMILTON |14|08|48|41|4D|49|4C|54|4F|4E|0D| ... |20|
3. ADAMS    |14|05|41|44|41|4D|53|0D|4F|4E|0D| ... |20|
```

Имя HAMILTON заменяет более короткое имя BROWN. Но, так как имя ADAMS короче имени HAMILTON, то оно заменяет только HAMIL. Код Return заменяет символ T. Остальные буквы - ON остаются после имени ADAMS. Для очистки поля NAMEFLD до ввода очередного имени может служить следующая программа:

```
    MOV  CX,20             ;Установить 20 циклов
    MOV  SI,0000           ;Начальная позиция поля
B30:
    MOV  NAMEFLD[si],20H   ;Переслать один пробел
    INC  SI                ;Следующая позиция поля
    LOOP B30               ;20 циклов
```

Вместо регистра SI можно использовать DI или BX. Более эффективный способ очистки поля, предполагающий пересылку слова из двух пробелов, требует только десять циклов. Однако, ввиду того что поле NAMEFLD определено как DB (байтовое), необходимо изменить длину в команде пересылки, посредством операнда WORD, а также воспользоваться операндом PTR (указатель), как показано ниже:

```
    MOV  CX,10             ;Установить 10 циклов
    LEA  SI,NAMEFLD        ;Начальный адрес
B30:
    MOV  WORD PTR[SI],2020H  ;Переслать два пробела
    INC  SI                ;Получить адрес
    INC  SI                ; следующего слова
```

```
        LOOP B30            ;10 циклов
```

Команда MOV по метке B30 обозначает пересылку слова из двух пробелов по адресу, находящемуся в регистре SI. В последнем примере используется команда LEA для инициализации регистра SI и несколько иной способ в команде MOV по метке B30, так как нельзя закодировать, например, следующую команду:

```
      MOV  WORD PTR[NAMEFLD],2020H  ;Неправильно
```

Очистка входной области решает проблему ввода коротких имен, за которыми следуют предыдущие данные. Еще более эффективный способ предполагает очистку только тех байт, которые расположены после введенного имени.

ЭКРАННЫЕ И КЛАВИАТУРНЫЕ ОПЕРАЦИИ: РАСШИРЕННАЯ ВЕРСИЯ DOS

Рассмотрим теперь расширенные возможности, введенные в DOS 2.0 (реализованные в стиле операционной системы UNIX). Если вы используете более младшую версию DOS, то не сможете выполнить примеры из данного раздела. Расширенные возможности включают файловый номер (file handle), который устанавливается в регистре BX, когда требуется выполнить операцию ввода/вывода. Существуют следующие стандартные файловые номера:

```
    0   Ввод (обычно с клавиатуры) CON
    1   Вывод (обычно на экран) CON
    2   Вывод по ошибке (на экран) CON
    3   Ввод/вывод на внешнее устройство AUX
    4   Вывод на печать LPT1 или PRN
```

Прерывание DOS для ввода/вывода - INT 21H, необходимая функция запрашивается через регистр AH: шест.3F - для ввода, шест.40 - для вывода. В регистр CX заносится число байт для ввода/вывода, а в регистр DX - адрес области ввода/вывода.

В результате успешного выполнения операции ввода/вывода очищается флаг переноса (CF) и в регистр AX устанавливается действительное число байт, участвующих в операции. При неуспешной операции устанавливается флаг CF, а код ошибки (в данном случае 6) заносится в регистр AX. Поскольку регистр AX может содержать как длину данных, так и код ошибки, то единственный способ определить наличие ошибки - проверить флаг CF, хотя ошибки чтения с клавиатуры и вывода на экран - явления крайне редкие. Аналогичным образом используются файловые номера для дисковых файлов, здесь ошибки ввода/вывода встречаются чаще.

Можно использовать эти функции для перенаправления ввода-вывода на другие устройства, однако эта особенность здесь не рассматривается.

ВЫВОД НА ЭКРАН: РАСШИРЕННАЯ ВЕРСИЯ DOS

Следующие команды иллюстрируют операцию вывода на экран в расширенной версии DOS:

```
      DISAREA   DB   20 DUP(' ')    ;Область данных
                ...
                MOV  AH,40H         ;Запрос на вывод
                MOV  BX,01          ;Выводное устройство
                MOV  CX,20          ;Максимальное число байт
                LEA  DX,DISAREA     ;Адрес области данных
```

```
            INT  21H          ;Вызов DOS
```

Команда LEA загружает в регистр DX адрес DISAREA для возможности DOS локализовать информацию, предназначенную для вывода. В результате успешной операции флаг переноса очищается (это можно проверить), а в регистре AX устанавливается число выведенных символов. Ошибка в данной операции может произойти, если установлен неправильный файловый номер. В этом случае будет установлен флаг CF и код ошибки (в данном случае 6) в регистре AX. Поскольку регистр AX может содержать или длину, или код ошибки, то единственный способ определить состояние ошибки - проверить флаг CF.

Упражнение: Вывод на экран

Воспользуемся отладчиком DEBUG для проверки внутренних эффектов прерывания. Загрузите DEBUG и после вывода на экран приглашения введите A 100 для ввода ассемблерных команд (не машинных команд) по адресу 100. Не забудьте, что DEBUG предполагает, что все числа вводятся в шестнадцатиричном формате:

```
100  MOV  AH,40
102  MOV  BX,01
105  MOV  CX,xx     (введите длину вашего имени)
108  MOV  DX,10E
10B  INT  21
10D  RET
10E  DB   'Ваше имя'
```

Программа устанавливает в регистре AH запрос на вывод и устанавливает шест. значение 10F в регистре DX - адрес DB, содержащей ваше имя в конце программы.

Когда вы наберете все команды, нажмите еще раз Return. С помощью команды U (U 100,10D) дисассемблируйте программу для проверки. Затем используйте команды R и T для трассировки выполнения. При выполнении команды INT 21H отладчик перейдет в BIOS, поэтому при достижении адреса 10B введите команду GO (G 10D) для перехода к команде RET. Ваше имя будет выведено на экран. С помощью команды Q вернитесь в DOS.

ВВОД С КЛАВИАТУРЫ: РАСШИРЕННЫЙ DOS

Ниже приведены команды, иллюстрирующие использование функции ввода с клавиатуры в расширенной версии DOS:

```
INAREA    DB   20 DUP (' ')   ;Область ввода
          MOV  AH,3FH         ;Запрос на ввод
          MOV  BX,00          ;Номер для клавиатуры
          MOV  CX,20          ;Максимум байт для ввода
          LEA  DX,INAREA      ;Адрес области ввода
          INT  21H            ;Вызов DOS
```

Команда LEA загружает относительный адрес INAREA в регистр DX. Команда INT ожидает, пока пользователь не введет символы с клавиатуры, но не проверяет превышает ли число введенных символов максимальное значение в регистре CX (20 в приведенном примере). Нажатие клавиши Return (код шест. 0D) указывает на завершение ввода. Например, после ввода текста "PC Users Group" INAREA будет содержать:

```
PC Users Group, шест.0D, шест.0A
```

После введенного текста непосредственно следует символ возврата каретки (шест.0D), который был введен, и символ конца строки (шест.0A), который не был введен. В силу данной особенности максимальное число символов и размер области ввода должны предусматривать место для двух символов. Если будет введено символов меньше максимального значения, то область памяти за введенными символами сохранит прежнее значение.

В результате успешной операции будет очищен флаг CF (что можно проверить) и в регистре AX будет установлено число байт, введенных с клавиатуры. В предыдущем примере это число будет равно 14 плюс 2 для перевода каретки и конца строки, т.е. 16. Соответствующим образом программа может определить действительное число введенных символов. Хотя данное свойство весьма тривиально для ответов типа YES или NO, оно может быть полезно для ответов с переменной длиной, таких, например, как имена.

Ошибка ввода может возникнуть, если определен неправильный номер файла. В этом случае будет установлен флаг CF и в регистр AX будет помещен код ошибки (6 в данном случае). Так как регистр AX может содержать или длину введенных данных, или код ошибки, то единственный способ определения наличия ошибки - проверка флага CF.

Если вводить текст, который превышает максимальную длину, установленную в регистре CX, то будут приниматься все символы. Рассмотрим ситуацию, когда регистр CX содержит 08,а пользователь введет символы "PC Exchange". В результате первые восемь символов "PC Excha" попадут в область ввода без кодов возврата каретки и конца строки. В регистре AX будет установлена длина 08. Следующая команда INT будет принимать данные не с клавиатуры, а из собственного буфера, поскольку там еще остались предыдущие данные. Таким образом, в область ввода будут приняты символы "nge", символ перевода каретки и символ новой строки, в регистре AX будет установлено значение 05. Обе операции ввода являются вполне нормальными и флаг CF будет очищен.

```
Первый INT:    PC Excha      AX = 08
Второй INT:    nge,0D,0A     AX = 05
```

Программа может определить факт ввода законченного текста, если а) в регистре AX получится значение меньше, чем в регистре CX или б) если содержимые AX и CX равны, но последние два символа в области ввода - 0D и 0A.

Встроенные в DOS проверки по функции 0AH для ввода с клавиатуры имеют более мощные средства. Их выбор для использования в программах является предпочтительным.

Упражнение: Ввод данных

Выполним упражнение в котором можно проследить операцию ввода с клавиатуры с помощью отладчика DEBUG. Предполагаемая программа позволяет вводить до 12 символов, включая символы конца каретки и конца строки. Загрузите DEBUG и после вывода на экран приглашения введите A 100 для ввода ассемблерных команд, начиная с адреса 100. Не забудьте, что DEBUG предполагает, что все числа вводятся в шестнадцатиричном формате.

```
100  MOV  AH,3F
102  MOV  BX,00
105  MOV  CX,0C
108  MOV  DX,10F
10B  INT  21
10D  JMP  100
10F  DB  ''
```

Программа устанавливает регистры AH и BX для запроса на ввод с клавиатуры, заносит максимальную длину ввода в регистр CX и загружает в

регистр DX значение 10F - область DB в конце программы. В эту область будут помещаться вводимые символы.

Когда вы наберете все команды, нажмите еще раз Return. С помощью команды U 100,108 выполните дисассемблирование программы для проверки. Затем используйте команды R и T для трассировки четырех команд MOV. Остановившись по адресу 10B, введите G 10D для выполнения команды INT (входить в BIOS не следует). Теперь отладчик позволит ввести данные, завершаемые клавишей Return. Проверьте содержимое регистра AX, состояние флага CF и используя команду D 10F, просмотрите введенные данные в памяти. Для завершения работы введите команду Q.

ИСПОЛЬЗОВАНИЕ СИМВОЛОВ ВОЗВРАТА КАРЕТКИ, КОНЦА СТРОКИ И ТАБУЛЯЦИИ ДЛЯ ВЫВОДА НА ЭКРАН

Один из способов получения более эффективного выводе на экран - использование управляющих символов возврата каретки, перевода строки и табуляции:

	Десятичные ASCII	Шестнадцатиричные
CR	13	0DH
LF	10	0AH
TAB	09	09H

Эти символы при операциях ввода-вывода выполняют одинаковые действия как в базовой, так и в расширенной версиях DOS. Например:

```
MESSAGE   DB       09,'PC Users Group Annual Report',13,10
          MOV      AH,40H          ;Запрос на вывод
          MOV      BX,01           ;Номер файла
          MOV      CX,31           ;Длина текста
          LEA      DX,MESSAGE      ;Адрес текста
          INT      21H             ;Вызов DOS
```

Использование директивы EQU для определения кодов делает программу более понятной:

```
CR           EQU      13             ;или EQU 0DH
LF           EQU      10             ;или EQU 0AH
TAB          EQU      09             ;или EQU 09H
MESSAGE DB     TAB,  'PC Users Group Annual' DB 'Report', CR, LF
```

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Команда INT 10H передает управление в BIOS для ввода с клавиатуры или вывода на экран. Команда INT 21H передает управление в DOS для выполнения некоторых более сложных действий при операциях ввода-вывода.

- Будьте внимательны при использовании шестнадцатиричной нотации, например INT 21 и INT 21H - это не одно и то же.

- Будьте внимательны при установке значений в регистры AX, BX, CX и DX для операций ввода-вывода.

- При использовании INT 21H в базовой версии DOS устанавливайте символ-ограничитель ($) непосредственно в конце области вывода. Будьте осторожны при очистке области - не удалите символ-ограничитель. Отсутствие

ограничителя может привести к непредвиденным эффектам на экране.

- Для ввода в базовой версии DOS тщательно определяйте список параметров. Экранные функции INT 21H предполагают, что первый байт содержит максимальную длину ввода, а второй байт заполняется системой автоматически значением действительной длины введенных данных.

- Для вывода на экран под управлением расширенной версии DOS устанавливайте в регистре AH значение 40H, а в регистре BX - файловый номер 01.

- Для ввода с клавиатуры под управлением расширенной версии DOS устанавливайте в регистре AH значение функции 3FH, а в регистре BX - файловый номер 00. Введенные в область ввода данные завершаются символами возврат каретки и перевод строки. Данная операция не контролирует ввод, превышающий по длине максимальное значение.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

8.1. Какие шестнадцатиричные значения строки и столбца соответствуют нижнему правому углу экрана 25x40?

8.2. Напишите команды для установки курсора по координатам: строка 12, столбец 8.

8.3. Напишите команды для очистки экрана, начиная с 0-го столбца 12-й строки до 79-го столбца 22-й строки.

8.4. Составьте необходимые элементы данных и команды для вывода запроса 'Введите дату (дд/мм/гг)'. За сообщением должен следовать звуковой сигнал. Используйте для вывода: а) функцию базовой версии DOS, б) функцию расширенной версии DOS и файловый номер.

8.5. Составьте необходимые элементы данных и команды для ввода с клавиатуры в формате вопроса 8.4. Используйте для ввода: а) функцию базовой версии DOS, б) функцию расширенной версии DOS и файловый номер.

8.6. Укажите стандартные файловые номера для ввода с клавиатуры, обычного вывода на экран и вывода на принтер.

8.7. Введите в компьютер программу из рис.8.2 со следующими изменениями, ассемблируйте ее, выполните компоновку и проверку работы: а) вместо строки 12 середина строки 15; б) вместо очистки всего экрана очистка только строк от 0 до 15.

8.8. Измените программу из рис.8.2 для использования ее в расширенной версии DOS. Выполните ее ассемблирование, компоновку и проверку работы.

ГЛАВА 9 Экранные операции II: Расширенные возможности

Ц е л ь: Показать более развитые возможности управления экраном, включая прокрутку, инвертирование, мигание, а также использование скэн-кодов для ввода с клавиатуры.

ВВЕДЕНИЕ

В гл.8 были показаны основные возможности системы для управления выводом на экран и ввода с клавиатуры. В данной главе приводятся более развитые возможности, обеспечивающие прокрутку данных на экране и установку байта-атрибута для подчеркивания, мигания, выделения яркости. Материал первого раздела этой главы (по прерыванию BIOS 10) подходит, как для монохромных, так и для цветных дисплеев. Другие расширенные возможности включают использование скэн-кодов для определения нажатой клавиши или комбинации клавишей на клавиатуре.

Монохромный дисплей

Для работы монохромного дисплея имеется память объемом 4К, начинающаяся по адресу шест. B0000 (дисплейный буфер). Эта память обеспечивает:

- 2К для символов на экране(25 строк x 80 столбцов);
- 2К для байтов-атрибутов, обеспечивающих инвертирование, мигание, выделение яркостью и подчеркивание.

Цветной/графический дисплей

Для работы стандартного цветного графического дисплея имеется 16 Кбайт памяти (дисплейный буфер), начинающийся по адресу шест.B8000. Такой дисплей может являться текстовым (для нормального ASCII-кода) или графическим и работать как в цветном, так и в черно-белом (BW) режиме. Дисплейный буфер обеспечивает экранные страницы, пронумерованные от 0 до 3 для экрана на 80 столбцов и от 0 до 7 для экрана на 40 столбцов. Номер страницы по умолчанию - 0. В следующей главе будет подробно рассмотрено управление цветом и графикой.

БАЙТ АТРИБУТОВ

Байт атрибутов, как для монохромного, так и для графического дисплея в текстовом (не графическом) режиме определяет характеристики каждого отображаемого символа. Байт-атрибут имеет следующие 8 бит:

```
                  Фон      Текст

     Атрибут:      BL  R  G  B    I  R  G  B
     Номер битов:   7   6  5  4    3  2  1  0
```

Буквы RGB представляют битовые позиции, управляющие красным (red), зеленым (green) и синим (blue) лучем в цветном мониторе. Бит 7 (BL) устанавливает мигание, а бит 3 (I) - уровень яркости. На монохромных мониторах текст высвечивается зеленым или оранжевым на темном фоне, хотя в данной главе такое изображение называется черно-белым (BW).

Для модификации атрибутов можно комбинировать биты следующим образом:

Эффект выделения	Фон RGB	Текст RGB
Неотображаемый (черный по черному)	000	000
Подчеркивание (не для цвета)	000	001
Нормальный (белый по черному)	000	111
Инвертированный (черный по белому)	111	000

Цветные мониторы не обеспечивают подчеркивания; вместо этого установка бит подчеркивания выбирает синий цвет для текста и получается отображение синим по черному. Ниже приведены некоторые атрибуты, основанные на комбинации битов фона, текста, мигания и выделения яркостью:

Двоичный код	Шест. код	Эффект выделения
0000 0000	00	Неотображаемый (для паролей)
0000 0111	07	Белый по черному (нормальный)
1000 0111	87	Белый по черному (мигание)
0000 1111	0F	Белый по черному (яркий)
0111 0000	70	Черный по белому (инвертированый)
1111 0000	F0	Черный по белому (инверт. мигающий)

Эти атрибуты подходят для текстового режима, как для монохромных, так и для цветных дисплеев. В следующей главе будет показано, как выбирать конкретные цвета. Для генерации атрибута можно использовать команду INT 10H. При этом регистр BL должен содержать значение байта-атрибута, а регистр AH один из следующих кодов: 06 (прокрутка вверх), 07 (прокрутка вниз), 08 (ввод атрибута или символа), 09 (вывод атрибута или символа). Если программа установила некоторый атрибут, то он остается таким, пока программа его не изменит. Если установить значение байта атрибута равным шест.00, то символ вообще не будет отображен.

ПРЕРЫВАНИЕ BIOS INT 10H

Прерывание INT 10H обеспечивает управление всем экраном. В регистре AH устанавливается код, определяющий функцию прерывания. Команда сохраняет содержимое регистров BX, CX, DX, SI и BP. Ниже описывается все возможные функции.

AH=00: Установка режима. Данная функция позволяет переключать цветной монитор в текстовый или графический режим. Установка режима для выполняемой в текущий момент программы осуществляется с помощью INT 10H. При установке происходит очистка экрана. Содержимое регистра AL может быть следующим:

00	40 x 25	черно-белый текстовый режим
01	40 x 25	стандартный i6-цветовой текстовый режим
02	80 x 25	черно-белый текстовый режим
03	80 x 25	стандартный 16-цветовой текстовый режим
04	320 x 200	стандартный 4-цветовой графический режим
05	320 x 200	черно-белый графический режим
06	640 x 200	черно-белый графический режим
07	80 x 25	черно-белый стандартный монохромный
08 - 0A		форматы для модели PCjr
0D	320 x 200	16-цветовой графический режим (EGA)
0E	640 x 200	16-цветовой графический режим (EGA)
0F	640 x 350	черно-белый графический режим (EGA)
10	640 x 350	64-цветовой графический режим (EGA)

EGA (Enhanced Graphics Adapter) - обозначает усовершенствованный графический адаптер. Следующий пример показывает установку стандартного 16-цветового текстового режима:

```
MOV  AH,00     ;Функция установки режима
MOV  AL,03     ;Стандартный цветной текст 80 x 25
INT  10H       ;Вызвать BIOS
```

Для определения типа адаптера, установленного в системе, служит прерывание BIOS INT 11H. Данная команда возвращает в регистре AX значение, в котором биты 5 и 4 указывают на видеорежим:

```
01     40 x 25   черно-белый режим в цветном адаптере
10     80 x 25   черно-белый режим в цветном адаптере
11     80 x 25   черно-белый режим в черно-белом адаптере
```

Программа, работающая с неизвестным типом монитора, может проверить тип по регистру AX после INT 11H и затем установить необходимый режим.

AH=01: Установка размера курсора. Курсор не является символом из набора ASCII-кодов. Компьютер имеет собственное аппаратное обеспечение для управления видом курсора. Для этого имеется специальная обработка по INT прерыванию. Обычно символ курсора похож на символ подчеркивания. Используя INT 10H, можно управлять вертикальным размером курсора: биты 4-0 в регистре CH для верхней линии сканирования, а биты 4-0 в регистре CL - для нижней. Можно установить любой размер курсора по вертикали: от 0 до 13 для монохромных и EGA мониторов и от 0 до 7 для большинства цветных мониторов. Приведем пример для увеличения размера курсора от его верхней до нижней линии сканирования:

```
MOV  AH,01     ;Установить размер курсора
MOV  CH,00     ;Верхняя линия сканирования
MOV  CL,13     ;Нижняя линия сканирования
INT  10H       ;Вызвать BIOS
```

В результате выполнения этих команд курсор превратится в сплошной мигающий прямоугольник. Можно установить любой размер курсора между верхней и нижней границами, например, 04/08, 03/10 и т.д. Курсор сохраняет свой вид, пока программа не изменит его. Использование размеров 12/13 (для моно) и 6/7 (для цвета) переводит курсор в его нормальный вид.

AH=02: Установка позиции курсора. Эта функция устанавливает курсор в любую позицию на экране в соответствии с координатами строки и столбца. Номер страницы обычно равен 0, но может иметь значение от 0 до 3 при 80 столбцах на экране. Для установки позиции курсора необходимо занести в регистр AH значение 02, в регистр BH номер страницы и в регистр DX координаты строки и столбца:

```
MOV  AH,02          ;Установить положение курсора
MOV  BH,00          ;Страница 0
MOV  DH,строка      ;Строка
MOV  DL,столбец     ;Столбец
INT  10H            ;Вызвать BIOS
```

AH=03: Чтение текущего положения курсора. Программа может определить положение курсора на экране (строку и столбец), а также размер курсора, следующим образом:

```
MOV  AH,03          ;Определить положение курсора
MOV  BH,00          ;Установить страницу 0
INT  10H            ;Вызвать BIOS
```

После возврата регистр DH будет содержать номер строки, а регистр DL - номер столбца. В регистре CH будет верхняя линия сканирования, а в регистре CL - нижняя.

AH=04: Чтение положения светового пера. Данная функция используется в графическом режиме для определения положения светового пера.

AH=05: Выбор активной страницы. Новая страница устанавливается для цветных текстовых режимов от 0 до 3. Для режима 40 x 25 возможно устанавливать до 8 страниц (от 0 до 7), а для режима 80 x 25 - до 4 страниц (от 0 до 3).

```
MOV  AH,05          ;Установить активную страницу
MOV  AL,страница    ;Номер страницы
INT  10H            ;Вызвать BIOS
```

AH=06: Прокрутка экрана вверх. Когда программа пытается выдать текст на строку ниже последней на экране, то происходит переход на верхнюю строку. Даже если с помощью прерывания будет специфицирован нулевой столбец, все равно предполагается новая строка, и нижние строки на экране будут испорчены. Для решения этой проблемы используется прокрутка экрана.

Ранее код 06 использовался для очистки экрана. В текстовом режиме установка в регистре AL значения 00 приводит к полной прокрутке вверх всего экрана, очищая его пробелами. Установка ненулевого значения в регистре AL определяет количество строк прокрутки экрана вверх. Верхние строки уходят с экрана, а чистые строки вводятся снизу. Следующие команды выполняют прокрутку всего экрана на одну строку:

```
MOV  AX,0601H  ;Прокрутить на одну строку вверх
MOV  BH,07     ;Атрибут: нормальный, черно-белый
MOV  CX,0000   ;Координаты от 00,00
MOV  DX,184FH  ; до 24,79 (полный экран)
INT  10H       ;Вызвать BIOS
```

Для прокрутки любого количества строк необходимо установить соответствующее значение в регистре AL. Регистр BH содержит атрибут для нормального или инвертированного отображения, мигания, установки цвета и т.д. Значения в регистрах CX и DX позволяют прокручивать любую часть экрана. Ниже объясняется стандартный подход к прокрутке:

1. Определить в элементе ROW (строка) значение 0 для установки строки положения курсора.
2. Выдать текст и продвинуть курсор на следующую строку.
3. Проверить, находится ли курсор на последней строке (CMP ROW,22).
4. Если да, то увеличить элемент ROW (INC ROW) и выйти.
5. Если нет, то прокрутить экран на одну строку и, используя ROW переустановить курсор.

AH=07: Прокрутка экрана вниз. Для текстового режима прокрутка экрана вниз обозначает удаление нижних строк и вставка чистых строк сверху. Регистр AH должен содержать 07, значения остальных регистров аналогичны функции 06 для прокрутки вверх.

AH=08: Чтение атрибута/символа в текущей позиции курсора. Для чтения символа и байта атрибута из дисплейного буфера, как в текстовом, так и в графическом режиме используются следующие команды:

```
MOV  AH,08     ;Запрос на чтение атр./симв.
MOV  BH,00     ;Страница 0 (для текстового реж.)
INT  10H       ;Вызвать BIOS
```

Данная функция возвращает в регистре AL значение символа, а в AH - его атрибут. В графическом режиме функция возвращает шест.00 для не ASCII-кодов. Так как эта функция читает только один символ, то для символьной строки необходима организация цикла.

AH=09: Вывод атрибута/символа в текущую позицию курсора. Для вывода на экран символов в текстовом или графическом режиме с установкой мигания, инвертирования и т.д. можно воспользоваться следующими командами:

```
MOV  AH,09          ;Функция вывода
MOV  AL,символ      ;Выводимый символ
MOV  BH,страница    ;Номер страницы (текст.реж.)
MOV  BL,атрибут     ;Атрибут или цвет
MOV  CX,повторение  ;Число повторений символа
INT  10H            ;Вызвать BIOS
```

В регистр AL должен быть помещен выводимый на экран символ. Значение в регистре CX определяет число повторений символа на экране. Вывод на экран последовательности различных символов требует организации цикла. Данная функция не перемещает курсор. В следующем примере на экран выводится пять мигающих "сердечек" в инвертированном виде:

```
MOV  AH,09          ;Функция вывода
MOV  AL,03H         ;Черви (карточная масть)
MOV  BH,00          ;Страница 0 (текст. режим)
MOV  BL,0F0H        ;Мигание, инверсия
MOV  CX,05          ;Пять раз
INT  10H            ;Вызвать BIOS
```

В текстовом (но не в графическом) режиме символы автоматически выводятся на экран и переходят с одной строки на другую. Для вывода на экран текста запроса или сообщения необходимо составить программу, которая устанавливает в регистре CX значение 01 и в цикле загружает в регистр AL из памяти выводимые символы текста. Так как регистр CX в данном случае занят, то нельзя использовать команду LOOP. Кроме того, при выводе каждого символа необходимо дополнительно продвигать курсор в следующий столбец (функция 02).

В графическом режиме регистр BL используется для определения цвета графики. Если бит 7 равен 0, то заданный цвет заменяет текущий цвет точки, если бит 7 равен 1, то происходит комбинация цветов с помощью команды XOR.

AH=0A: Вывод символа в текущую позицию курсора. Единственная разница между функциями 0A и 09 состоит в том, что функция 0A не устанавливает атрибут:

```
MOV  AH,0AH         ;Функция вывода
MOV  AL,символ      ;Выводимый символ
MOV  BH,страница    ;Номер страницы (для текста)
MOV  CX,повторение  ;Число повторений символа
INT  10H            ;Вызвать BIOS
```

Для большинства применений команда прерывания DOS INT 21H более удобна.

AH=0E: Вывод в режиме телетайпа. Данная функция позволяет использовать монитор, как простой терминал. Для выполнения этой функции необходимо установить в регистре AH шест. значение 0E, в регистр AL поместить выводимый символ, цвет текста (в графическом режиме) занести в регистр BL и номер страницы для текстового режима - в регистр BH. Звуковой сигнал (код 07H), возврат на одну позицию (08H), конец строки (0AH) и возврат каретки (0DH) действуют, как команды для форматизации экрана. Данная функция автоматически продвигает курсор, переводит символы на следующую строку, выполняет прокрутку экрана и сохраняет текущие атрибуты экрана.

AH=0F: Получение текущего видео режима. Данная функция возвращает в

регистре AL текущий видео режим (см.функцию AH=00), в регистре AH - число символов в строке (20, 40 или 80), в регистре BH - номер страницы.

AH=13: Вывод символьной строки (только для AT). Данная функция позволяет на компьютерах типа AT выводить на экран символьные строки с установкой атрибутов и перемещением курсора:

```
MOV  AH,13H         ;Функция вывода на экран
MOV  AL,сервис      ;0, 1, 2 или 3
MOV  BH,страница    ;
LEA  BP,адрес       ;Адрес строки в ES:BP
MOV  CX,длина       ;Длина строки
MOV  DX,экран       ;Координаты на экране
INT  10H            ;Вызвать BIOS
```

Возможен следующий дополнительный сервис:
0 - использовать атрибут и не перемещать курсор;
1 - использовать атрибут и переместить курсор;
2 - вывести символ, затем атрибут и не перемещать курсор;
3 - вывести символ, затем атрибут и переместить курсор.

ПРОГРАММА: МИГАНИЕ, ИНВЕРСИЯ И ПРОКРУТКА

Программа, приведенная на рис. 9.1, принимает ввод имен с клавиатуры и выводит их на экран. Запрос выдается в инвертированном отображении, имена принимаются в нормальном отображении, а вывод имен осуществляется с 40 столбца в той же строке с миганием и инвертированием:

```
Name? Francis Bacon      Francis Bacon [мигание]
|                        |
Столбец 0                Столбец 40
```

Для управления положением курсора в программе определены переменные ROW (вертикальное перемещение вниз) и COL (горизонтальное перемещение вправо). Команда INT 10H не перемещает курсор автоматически. Программа выводит имена сверху вниз, пока не достигнет 20-й строки. После этого выполняется прокрутка экрана вверх на одну строку для каждого нового запроса.

```
       page     60,132
TITLE   NMSCROLL (EXE) Инвертирование, мигание, прокрутка
; ------------------------------------------------------
STACKSG SEGMENT PARA STACK 'Stack'
       DW      32 DUP(?)
STACKSG SEGMENT PARA STACK 'Stack'
       DW      32 DUP(?)
STACKG  ENDS
; ------------------------------------------------------
DATASG  SEGMENT PARA  'Data'
NAMEPAR LABEL   BYTE           ;Имя списка параметров:
MAXNLEN DB      20             ; макс. длина имени
ACTNLEN DB      ?              ; число введенных символов
NAMEFLD DB      20 DUP(' ')    ; имя

COL     DB      00
COUNT   DB      ?
PROMPT  DB      'Name? '
```

```
ROW     DB      00
DATASG  ENDS
; -------------------------------------------------------
CODESG  SEGMENT PARA  'Code'
BEGIN   PROC    FAR
        ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASG
        PUSH    DS
        SUB     AX,AX
        PUSH    DS
        MOV     AX,DATASG
        MOV     ES,AX
        MOV     AX,0600H
        CALL    Q10CLR          ;Очистить экран
A20LOOP:
        MOV     COL,00          ;Установить столбец 0
        CALL    Q20CURS
        CALL    B10PRMP         ;Выдать текст запроса
        CALL    D10INPT         ;Ввести имя с клавиатуры
        CMP     ACTNLEN,00      ;Нет имени? (т.е. конец)
        JNE     A30
        MOV     AX,0600H
        CALL    Q10CLR          ;Если да, то очистить экран,
        RET                     ; и завершить программу
A30:
        CALL    E10NAME         ;Вывести имя на экран
        JMP     A20LOOP
BEGIN   ENDP
;               Вывод текста запроса:
;               --------------------
B10PRMP PROC    NEAR
        LEA     SI,PROMPT       ;Адрес текста
        MOV     COUNT,05
B20:
        MOV     BL,70H          ;Видеоинверсия
        CALL    F10DISP         ;Подпрограмма вывода
        INC     SI              ;Следующий символ в имени
        INC     COL             ;Следующий столбец
        CALL    Q20CURS
        DEC     COUNT           ;Уменьшение счетчика
        JNZ     B20             ;Повторить n раз
        RET
B10PRMP ENDP
;               вод имени с клавиатуры:
;               ----------------------
D10INPT PROC    NEAR
        MOV     AN,0AH
        LEA     DX,NAMEPAR
        INT     21H
        RET
D10INPT ENDP
;               Вывод имени с миганием и инверсией:
;               ---------------------------------
E10NAME PROC    NEAR
        LEA     SI,NAMEFLD      ;Адрес имени
        MOV     COL,40          ;Установить столбец
E20:
        CALL    Q20CURS         ;Установить курсор
        MOV     BL,0FOH         ;Мигание и инверсия
        CALL    F10DISP         ;Подпрограмма вывода
        INS     SI              ;Следующий символ в имени
        INS     COL             ;Следующий столбец на экране
```

```
       DES     ACTNLEN         ;Уменьшить счетчик длины
       JNZ     E20             ;Циклить n раз
       CMP     ROW,20          ;Последняя строка экрана?
       JAE     E30             ; нет
       INC     ROW
       RET
E30:   MOV     AX,0601H        ; да --
       CALL    Q10CLR          ; очистить экран
       RET
E10NAME ENDP
;              Вывод символа на экран:
;              ----------------------
F10DISP PROC    NEAR            ;BL (атрибут) уже установлен
       MOV     AN,09           ;Функция вывода на экран
       MOV     AL,[SI]         ;Получить символ из имени
       MOV     BH,00           ;Номер страницы
       MOV     CX,01           ;Один символ
       INT     10H             ;Вызов BIOS
       RET
F10DISP ENDP
;              Очистка экрана:
;              --------------
Q10CLR  PROC    NEAR            ;AX установлен при вызове
       MOV     BH,07           ;Нормальный ч/б
       MOV     CX 0000
       MOV     DX,184FH
       INT     10H             ;Вызов BIOS
       RET
Q10CLR  ENDP
;              Установить курсор (строка/столбец):
;              ---------------------------------
Q20CURS PROC    NEAR
       MOV     AN,02
       MOV     BH,00
       MOV     DH,ROW
       MOV     DL,COL
       INT     10H
       RET
Q20CURS ENDP
CODESG  ENDS
       END     BEGIN
```

Рис.9.1. Мигание, инвертирование и прокрутка.

Для ввода имен в процедуре D10INPT используется команда DOS INT 21H. Для замены на BIOS INT 10H необходимо:

1. Инициализировать счетчик для адреса области ввода и счетчик для длины имени.
2. Выполнить INT 10H (функция 08) с 08 в регистре AH и 00 в BH. Функция возвращает каждый символ в регистре AL.
3. Если регистр AL не содержит символа RETURN и счетчик длины достиг максимального значения, выдать звуковой сигнал и выйти из процедуры.
4. Переслать содержимое AL в область ввода имени.
5. Если регистр AL содержит символ RETURN, выйти из процедуры.
6. Увеличить счетчик длины и адрес области ввода имени.
7. Переместить курсор на один столбец.
8. Перейти на пункт 2.

При выходе из процедуры область ввода содержит имя и символ RETURN, а счетчик - число введенных символов.

РАСШИРЕННЫЙ ASCII КОД

ASCII-коды от 128 до 255 (шест. 80-FF) представляют собой ряд специальных символов полезных при формировании запросов, меню, специальных значков с экранными атрибутами. Например, используя следующие символы можно нарисовать прямоугольник:

Шест.	Символ
DA	Верхний левый угол
BF	Верхний правый угол
C0	Нижний левый угол
D9	Нижний правый угол
C4	Горизонтальная линия
B3	Вертикальная линия

Следующие команды с помощью INT 10H выводят горизонтальную линию на 25 позиций в длину:

```
MOV  AH,09         ;Функция вывода на экран
MOV  AL,0C4H       ;Горизонтальная линия
MOV  BH,00         ;Страница 0
MOV  BL,0FH        ;Выделение яркостью
MOV  CX,25         ;25 повторений
MOV  10H           ;Вызвать BIOS
```

Напомним, что курсор не перемещается. Вывод вертикальной линии включает цикл, в котором курсор перемещается вниз на одну строку и выводится символ шест. B3. Для штриховки может быть полезен символ с точками внутри:

Шест.	Символ
B0	Одна четверть точек (светлая штриховка)
B1	Половина точек (средняя штриховка)
B2	Три четверти точек (темная штриховка)

Можно извлечь много полезных идей, изучая программное обеспечение с профессионально организованным выводом, или самому изобрести оригинальные идеи для отображения информации.

ДРУГИЕ ОПЕРАЦИИ ВВОДА/ВЫВОДА В DOS

Ниже перечислены другие функции DOS, которые могут оказаться полезными в работе. Код функции устанавливается в регистре AH и, затем, выдается команда INT 21H.

AH=01: Ввод с клавиатуры с эхо отображением. Данная функция возвращает значение в регистре AL. Если содержимое AL не равно нулю, то оно представляет собой стандартный ASCII- символ, например, букву или цифру. Нулевое значение в регистре AL свидетельствует о том, что на клавиатуре была нажата специальная функциональная клавиша, например, Home, F1 или PgUp. Для определения скэн-кода клавиш, необходимо повторить вызов

функции (см. "Дополнительные функциональные клавиши" в последующих разделах). Данная функция реагирует на запрос Ctrl/Break.

AH=02: Вывод символа. Для вывода символа на экран в текущую позицию курсора необходимо поместить код данного символа в регистр DL. Коды табуляции, возврата каретки и конца строки действуют обычным образом.

AH=07: Прямой ввод с клавиатуры без эхо отображения. Данная функция работает аналогично функции 01 с двумя отличия ми: введенный символ не отображается на экране, т.е. нет эхо, и отсутствует реакция на запрос Ctrl/Break.

AH=08: Ввод с клавиатуры без эхо отображения. Данная функция действует аналогично функции 01 с одним отличием: введенный символ не отображается на экран, т.е. нет эхо.

AH=0B: Проверка состояния клавиатуры. Данная функция возвращает шест. FF в регистре AL, если ввод с клавиатуры возможен, в противном случае - 00. Это средство связано с функциями 01, 07 и 08, которые не ожидают ввода с клавиатуры.

ВВОД С КЛАВИАТУРЫ ПО КОМАНДЕ BIOS INT 16H

Команда BIOS INT 16H выполняет специальную операцию, которая в соответствии с кодом в регистре AH обеспечивает следующие три функции ввода с клавиатуры.

AH=00: Чтение символа. Данная функция помещает в регистр AL очередной ASCII символ, введенный с клавиатуры, и устанавливает скэн-код в регистре AH. (Скэн-коды объясняются в следующем разделе). Если на клавиатуре нажата одна из специальных клавишей, например, Home или F1, то в регистр AL заносится 00. Автоматическое эхо символа на экран по этой функции не происходит.

AH=01: Определение наличия введенного символа. Данная функция сбрасывает флаг нуля (ZF=0), если имеется символ для чтения с клавиатуры; очередной символ и скэн-код будут помещены в регистры AL и AH соответственно и данный элемент останется в буфере.

AH=02: Определение текущего состояния клавиатуры. Данная функция возвращает в регистре AL состояние клавиатуры из адреса памяти шест.417:

Бит
 7 Состояние вставки активно (Ins)
 6 Состояние фиксации верхнего регистра (Caps Lock) переключено
 5 Состояние фиксации цифровой клавиатуры (Num Lock) переключено
 4 Состояние фиксации прокрутки (Scroll Lock) переключено
 3 Нажата комбинация клавишей Alt/Shift
 2 Нажата комбинация клавишей Ctrl/Shift
 1 Нажата левая клавиша Shift
 0 Нажата правая клавиша Shift

ФУНКЦИОНАЛЬНЫЕ КЛАВИШИ

Клавиатура располагает тремя основными типами клавишей:

1. Символьные (алфавитно-цифровые) клавиши: буквы от a до z,

цифры от 0 до 9, символы %, $, # и т.д.

2. Функциональные клавиши: Home, End, Возврат на позицию, стрелки, Return, Del, Ins, PgUp, PgDn и программно-функциональные клавиши.

3. Управляющие клавиши: Alt, Ctrl и Shift, которые работают совместно с другими клавишами.

Функциональная клавиша не вырабатывает какой-либо символ, но чаще формирует запрос на некоторые действия. Аппаратная реализация не требует от функциональных клавишей выполнения каких-либо специфических действий. Задачей программиста является определить, например, что нажатие клавиши Home должно привести к установке курсора в верхний левый угол экрана, или нажатие клавиши End должно установить курсор в конец текста на экране. Можно легко запрограммировать функциональные клавиши для выполнения самых различных действий.

Каждая клавиша имеет собственный скэн-код от 1 (Esc) до 83 (Del) или от шест.01 до шест.53. Посредством этих скэн-кодов программа может определить нажатие любой клавиши. Например, запрос на ввод одного символа с клавиатуры включает загрузку 00 в регистр AH и обращение к BIOS через INT 16H:

```
MOV  AH,00     ;Функция ввода с клавиатуры
INT  16H       ;Вызвать BIOS
```

Данная операция имеет два типа ответов в зависимости от того, нажата символьная клавиша или функциональная. Для символа (например, буква A) клавиатура посылает в компьютер два элемента информации:

1. ASCII-код символа A (шест.41) в регистре AL;
2. Скэн-код для клавиши A (шест.1E) в регистре AH.

Если нажата функциональная клавиша (например, Ins) клавиатура также передает два элемента:

1. Нуль в регистре AL;
2. Скэн-код для клавиши Ins (шест.52) в регистре AH.

Таким образом, после выполнения команды INT 16H необходимо прежде проверить содержимое регистра AL. Если AL содержит нуль, то была нажата функциональная клавиша, если не нуль, то получен код символьной клавиши. Ниже приведен пример такой проверки:

```
MOV  AH,00     ;Функция ввода
INT  16H       ;Вызвать BIOS
CMP  AL,00     ;Функциональная клавиша?
JZ   exit      ; да - выйти
```

Скэн-коды

На рис.9.2 приведены скэн-коды для некоторых функциональных клавишей. Клавиатура имеет по две клавиши для таких символов как *, + и -. Нажатие "звездочки", например, устанавливает код символа шест.2A в регистре AL и один из двух скэн-кодов в регистре AH в зависимости от того, какая из клавишей была нажата: шест.09 для звездочки над цифрой 8 или шест.29 для звездочки на клавише PrtSc.

Ниже приведена логика проверки скэн-кода для звездочки:

```
CMP  AL,2AH    ;Звездочка?
JNE  EXIT1     ; нет - выйти
CMP  AH,09H    ;Какой скэн-код?
JE   EXIT2
```

```
Функциональные клавиши          Скэн-коды

  Alt/A - Alt/Z                 1E - 2C
  F1 - F10                      3B - 44
  Home                          47
  Стрелка вверх                 48
  PgUp                          49
  Стрелка влево                 4B
  Стрелка вправо                4D
  End                           4F
  Стрелка вниз                  50
  PgDn                          51
  Ins                           52
  Del                           53
```

Рис.9.2. Скэн-коды некоторых функциональных клавишей

Приведем пример программы для установки курсора в строку 0 и столбец 0 при нажатии клавиши Home (скэн-код 47):

```
MOV  AH,00     ;Выполнить ввод с клавиатуры
INT  16H       ;
CMP  AL,00     ;Функциональная клавиша?
JNE  EXIT1     ; нет -- выйти
CMP  AH,47H    ;Скэн-код для клавиши Home?
JNE  EXIT2     ; нет -- выйти
MOV  AH,02     ;
MOV  BH,00     ;Установить курсор
MOV  DX,00     ; по координатам 0,0
INT  10H       ;Вызвать BIOS
```

Функциональные клавиши F1 - F10 генерируют скэн-коды от шест.3B до шест.44. Следующий пример выполняет проверку на функциональную клавишу F10:

```
CMP  AH,44H    ;Клавиша F10?
JE   EXIT1     ; Да!
```

По адресу EXIT1 программа может выполнить любое необходимое действие.

Полный список скэн-кодов приводится в руководстве по языку BASIC. Техническое описание IBM PC содержит подробное описание всех скэн-кодов, а также описание использования клавишей Alt, Ctrl и Shift.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Монохромный дисплей использует 4K байт памяти, 2K байт на символы и 2K байт на атрибуты для каждого символа.

- Цветной дисплей использует 16K байт памяти и может работать в цветном или черно-белом (BW) режимах. Возможно использование, как текстового режима для отображения ASCII-символов, так и графического режима для любых изображений.

- Байт-атрибут используется и для монохромных дисплеев и для цветных в текстовом режиме. Атрибут обеспечивает мигание, инвертирование и

выделение яркостью. Для цветных дисплеев в текстовом режиме биты RGB позволяют выбирать цвета, но не имеют режима подчеркивания.

- Команда BIOS INT 10H обеспечивает полную экранную обработку: установку режимов, установку положения курсора, прокрутку экрана, чтение с клавиатуры и вывод на экран.

- Если ваша программа выводит вниз экрана, то не забывай те выполнять прокрутку прежде, чем курсор выйдет из последней строки.

- При использовании атрибутов для мигания и инвертирования, не забывайте сбрасывать их в отключенное состояние.

- Для функций по команде INT 10H, выполняющих чтение и вывод на экран, помните о перемещении курсора.

- Команда BIOS INT 16H обеспечивает прием и распознавание функциональных клавишей.

- Функциональные клавиши предполагают запрограммированный вызов некоторых действий.

- Каждая клавиша на клавиатуре имеет конкретный скэн-код, пронумерованный от 1 (Esc) до 83 (Del), или от шест.01 до шест.53.

- Нажатие символьной клавиши на клавиатуре передает код символа в регистр AL и скэн-код клавиши в регистр AH.

- Нажатие функциональной клавиши на клавиатуре передает нуль в регистр AL и скэн-код клавиши в регистр AH.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

9.1. Определите атрибуты экрана для а) мигания с подчеркиванием, б) нормальной яркости, в) инвертирования с выделением яркостью.

9.2. Составьте процедуры для а) установки режима экрана BW (черно-белый) на 80 столбцов, б) установки вида курсора, начинающегося на 5 линии сканирования и заканчивающегося на 12 линии, в) прокрутки экрана на 10 строк, г) вывода десяти мигающих символов штриховки с половиной точек (шест. B1).

9.3. Напишите скэн-коды для следующих функциональных клавишей: а) стрелка вверх, б) клавиша F3, в) Home, г) PgUp.

9.4. Используя отладчик DEBUG, проверьте воздействие на содержимое регистра AX при нажатии клавишей на клавиатуре. Для ввода ассемблерных команд используйте команду A 100 (Return). Ведите следующие команды:

```
MOV  AH,00
INT  16H
JMP  100
```

Используя команду U 100,104, дисассемблируйте программу и с помощью G 104 выполните команды MOV и INT. На команде INT выполнение программы остановиться и система перейдет в ожидание вашего ввода. Для проверки регистра AH нажмите любую клавишу. Продолжая вводить команду G 104, и, нажимая различные клавиши, проверьте работу программы. Для выхода введите команду Q.

9.5. Составьте команды для определения нажатия клавиши: если нажата клавиша PgDn, то необходимо установить курсор по координатам - строка 24 и столбец 0.

ГЛАВА 10 Экранные операции III: Цвет и графика

Ц е л ь: Показать расширенные возможности компьютера, связанные с использованием цвета и графики на экране.

ВВЕДЕНИЕ

Данная глава знакомит с использованием цвета для текстового и графического режимов. Существуют следующие три типа видео мониторов, используемые для изображения цветной графики (в порядке возрастания стоимости и качества):

1. Немодифицированный цветной телевизионный приемник (обычный домашний телевизор), применяемый многими для своих компьютеров.
2. Комбинированный видеомонитор, принимающий цветовой сигнал без радиочастотной модуляции, и используемый для передачи по радиоволнам. Обеспечивает высокое качество изображения.
3. RGB-монитор, посылающий входные сигналы на три раздельные электронные пушки - красную, зеленую и синюю для каждого из трех основных цветов. Являясь наиболее дорогим, RGB-монитор обеспечивает наилучшее качество изображения.

Стандартный адаптер для цветного графического монитора (CGA - Color/Graphics Adapter) использует 16К байт памяти, начинающейся по адресу шест.B8000, 8К байт - для символов и 8К байт для их атрибутов. При работе в формате 80x25 адаптер может хранить четыре страницы (0-3) дисплейного буфера по 4К байт каждая. При работе в формате 40x25 адаптер может хранить восемь страниц (0-7) по 2К байт каждая. По умолчанию используется нулевая страница (в начале дисплейной памяти). Программа может вывести на экран любую страницу и в это время формировать другую страницу в памяти для последующего вывода на экран.

Усовершенствованный графический адаптер (EGA - Enhanced Graphics Adapter) обеспечивает более высокую разрешающую способность, по сравнению со стандартным цветным адаптером (CGA) и в большинстве случаев является совместимым с ним. Разрешающая способность обеспечивает 320x200, 640x200 и 640x350 точек на экране.

Цветные адаптеры имеют два основных режима работы: текстовой (алфавитно-цифровой) и графический, и возможны также дополнительные режимы между двумя основными. По умолчанию используется текстовой режим. Установка режима описана в главе 9 в разделе "Прерывание BIOS INT 10H" (AH=0). Для установки графического режима или возврата в текстовой режим используется прерывание BIOS INT 10H, как это показано в двух следующих примерах:

```
        MOV  AH,00  ;Режим       MOV  AH,00  ;Режим
        MOV  AL,03  ;Цвет+текст  MOV  AL,04  ;Графика среднего
        INT  10H                 INT 10H     ; разрешения
```

ТЕКСТОВЫЙ (АЛФАВИТНО-ЦИФРОВОЙ) РЕЖИМ

Текстовой режим предназначен для обычных вычислений с выводом букв и цифр на экран. Данный режим одинаков для черно-белых (BW) и для цветных мониторов за исключением того, что цветные мониторы не поддерживают атрибут подчеркивания. Текстовой режим обеспечивает работу с полным набором ASCII кодов (256 символов), как для черно-белых (BW), так и для цветных мониторов. Каждый символ на экране может отображаться в одном из 16 цветов на одном из восьми цветов фона. Бордюр экрана может иметь также один из 16 цветов.

Цвета

Тремя основными цветами являются красный, зеленый и синий. Комбинируя основные цвета, друг с другом, можно получить восемь цветов, включая черный и белый. Используя два уровня яркости для каждого цвета, получим всего 16 цветов:

	I R G B		I R G B
Черный	0 0 0 0	Серый	1 0 0 0
Синий	0 0 0 1	Ярко-синий	1 0 0 1
Зеленый	0 0 1 0	Ярко-зеленый	1 0 1 0
Голубой	0 0 1 1	Ярко-голубой	1 0 1 1
Красный	0 1 0 0	Ярко-красный	1 1 0 0
Сиреневый	0 1 0 1	Ярко-сиреневый	1 1 0 1
Коричневый	0 1 1 0	Желтый	1 1 1 0
Белый	0 1 1 0	Ярко-белый	1 1 1 1

Таким образом любые символы могут быть отображены на экране в одном из 16 цветов. Фон любого символа может иметь один из первых восьми цветов. Если фон и текст имеют один и тот же цвет, то текст получается невидимым. Используя байт атрибута, можно получить также мигающие символы.

Байт-атрибут

Текстовой режим допускает использование байта атрибута, рассмотренного в гл.9. В приведенной ниже таблице, атрибут BL обозначает мигание (BLinking), RGB - соответственно красный, зеленый и синий цвет, I - выделение яркостью:

	Фон	Текст
Атрибут:	BL R G B	I R G B
Номера битов:	7 6 5 4	3 2 1 0

Мигание и выделение яркостью относится к тексту. Ниже приведены некоторые типичные атрибуты:

Текст по фону	Бит: 7	6	5	4	3	2	1	0	
	BL	R	G	B	I	R	G	B	Шест.
Черный по черному	0	0	0	0	0	0	0	0	00
Синий по черному	0	0	0	0	0	0	0	1	01
Красный по синему	0	0	0	1	0	1	0	0	14
Голубой по зеленому	0	0	1	0	0	0	1	1	23
Светло-сиреневый по белому	0	1	1	1	1	1	0	1	7D
Серый по зеленому, мигание	1	0	1	0	1	0	0	0	A8

Байт-атрибут используется аналагично показанному для черно-белого (BW) монитора. Тип монитора можно определить из программы с помощью команды INT 11H. Для BW монитора код 07 устанавливает нормальный атрибут.

Для цветных мониторов можно использовать любую из цветовых комбинаций. Цвет на экране сохраняется до тех пор, пока другая команда не изменит его. Для установки цвета можно использовать в команде INT 10H функции AH=06, AH=07 и AH=09. Например, для вывода пяти мигающих звездочек светло-зеленым цветом на сиреневом фоне возможна следующая программа:

```
MOV  AH,09        ;Функция вывода на экран
MOV  AL,'*'       ;Выводимый символ
MOV  BH,00        ;Страница 0
MOV  BL,0DAH      ;Атрибут цвета
MOV  CX,05        ;Число повторений
INT  10H          ;Вызвать BIOS
```

ГРАФИЧЕСКИЙ РЕЖИМ

Для генерации цветных изображений в графическом режиме используются минимальные точки растра - пиксели или пэлы (pixel). Цветной графический адаптер (CGA) имеет три степени разрешения:

1. Низкое разрешение (не поддерживается в ROM) обеспечивает вывод 100 строк по 160 точек (т.е. четыре бита на точку). Каждая точка может иметь один из 16 стандартных цветов, как описано в предыдущем разделе "Цвета". Реализация данного режима включает прямую адресацию контролера Motorola 6845 CRT. Для этого используются два порта: шест.3D4 и 3D5.
2. Среднее разрешение для стандартной цветной графики обеспечивает 200 строк по 320 точек. Каждый байт в этом случае представляет четыре точки (т.е. два бита на точку).
3. Высокое разрешение обеспечивает 200 строк по 640 точек. Поскольку в данном случае требуется 16К байт памяти, высокое разрешение достигается только в черно-белом (BW) режиме. Каждый байт здесь представляет 8 точек (т.е. один бит на точку). Нулевое значение бита дает черный цвет точки, единичное - белый.

Заметим, что в графическом режиме ROM содержит точечные образы только для первых 128 ASCII-кодов. Команда INT 1FH обеспечивает доступ к 1К байтовой области в памяти, определяющей остальные 128 символов. (8 байт на символ). Отображение графических байтов в видео сигналы аналогично, как для среднего, так и для высокого разрешения.

РЕЖИМ СРЕДНЕГО РАЗРЕШЕНИЯ

При среднем разрешении каждый байт представляет четыре точки, пронумерованных от 0 до 3:

```
Байт:     |C1 C0|C1 C0|C1 C0|C1 C0|
Пиксели:     0     1     2     3
```

В любой момент для каждой точки возможны четыре цвета, от 0 до 3. Ограничение в 4 цвета объясняется тем, что двухбитовая точка имеет 4 комбинации значений битов: 00, 01, 10 и 11. Можно выбрать значение 00 для любого из 16 возможных цветов фона или выбрать значение 01, 10, и 11 для одной из двух палитр. Каждая палитра имеет три цвета:

```
C1   C0       Палитра 0     Палитра 1

0    0        фон           фон
```

```
0   1       зеленый       голубой
1   0       красный       сиреневый
1   1       коричневый     белый
```

Для выбора цвета палитры и фона используется INT 10H. Таким образом, если, например, выбран фон желтого цвета и палитра 0, то возможны следующие цвета точки: желтый, зеленый, красный и коричневый. Байт, содержащий значение 10101010, соответствует красным точкам. Если выбрать цвет фона - синий и палитру 1, то возможные цвета: синий, голубой, сиреневый и белый. Байт, содержащий значение 00011011, отображает синюю, голубую, сиреневую и белую точки.

Прерывание BIOS INT 10H для графики

Функция AH=00 команды INT 10H устанавливает графический режим. Функция AH=11 команды INT 10H позволяет выбрать цвет палитры и вывести на экран графический символ. Код в регистре AH определяет функцию:

AH=00: Установка режима. Нулевое значение в регистре AH и 04 в регистре AL устанавливают стандартный цветной графический режим:

```
MOV  AH,00     ;Функция установки режима
MOV  AL,04     ;Разрешение 320x200
INT  10H
```

Установка графического режима приводит к исчезновению курсора с экрана. Подробности по установке режима приведены в главе 9.

AH=0BH: Установка цветовой палитры. Число в регистре BH определяет назначение регистра BL:

BH=00 выбирает цвета фона и бордюра в соответствии с содержимым регистра BL. Цвет фона от 1 до 16 соответствует шест. значениям от 0 до F;

BH=01 выбирает палитру соответственно содержимому регистра BL (0 или 1):

```
MOV  AH,0BH    ;Функция установки цвета
MOV  BH,01     ;Выбор палитры
MOV  BL,00     ; 0 (зеленый, красный, корич.)
INT  10H       ;Вызвать BIOS
```

Палитра, установленная один раз, сохраняется, пока не будет отменена другой командой. При смене палитры весь экран меняет цветовую комбинацию. При использовании функции AH=0BH в текстовом режиме, значение, установленное для цвета 0 в палитре, определяет цвет бордюра.

AH=0CH: Вывод точки на экран. Использование кода 0C в регистре AH позволяет вывести на экран точку в выбранном цвете (фон и палитра). Например, для разрешения 320x200 загрузим в регистр DX вертикальную координату (от 0 до 199), а в регистр CX - горизонтальную координату (от 0 до 319). В регистр AL поместим цвет точки (от 0 до 3):

```
MOV  AH,0CH         ;Функция вывода точки
MOV  AL,цвет        ;Цвет точки
MOV  CX,столбец     ;Горизонтальная координата
MOV  DX,строка      ;Вертикальная координата
INT  10H            ;Вызвать BIOS
```

AH=0DH: Чтение точки с экрана. Данная функция позволяет прочитать

точку для определения ее цвета. В регистр DX должна быть загружена вертикальная координата (от 0 до 199), а в регистр CX - горизонтальная (от 0 до 319). В регистре AH должно быть значение 0D. Функция возвращает цвет точки в регистре AL.

ПРОГРАММА: УСТАНОВКА ГРАФИЧЕСКОГО РЕЖИМА И ОТОБРАЖЕНИЕ ЦВЕТА

Программа, приведенная на рис.10.1, использует команду INT 10H для установки графического режима, выбора зеленого фона и вывода на экран точек (40 строк по 320 столбцов). В программе происходит увеличение значения цвета на 1 для каждой строки. Так как в определении цвета участвуют только три правых бита, цвета повторяются через каждые семь строк.

```
TITLE   GRAPHIX (COM) Пример цвета и графики
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
        ORG     100H

MAIN    PROC    NEAR
        MOV     AN,00           ;Установка режима графики
        MOV     AL,0DH          ; для EGA (CGA=04)
        MOV     AH,0BH          ;Установить палитру
        MOV     BH,00           ;Фон
        MOV     BL,02           ;Зеленый
        INT     10H
        MOV     BX,00           ;Начальные цвет,
        MOV     CX,00           ; столбец
        MOV     DX,00           ; и строка
A50:
        MOV     AH,0CH          ;Функция вывода точки
        MOV     AL,BL           ;Установить цвет
        INT     10H             ;BX, CX, и DX сохраняются
        INC     CX              ;Увеличить столбец
        CMP     CX,320          ;Столбец 320?
        JNE     A50             ; нет - цикл,
        MOV     CX,00           ; да - сбросить столбец
        INS     BL              ;Изменить цвет
        INS     DX              ;Увеличить строку
        CMP     DX,40           ;Строка 40?
        JNE     A50             ; нет - цикл,
        RET                     ; да - завершить
MAIN    ENDP
CODESG  ENDS
        END     MAIN
```

Рис.10.1 Вывод на экран в цветном графическом режиме.

После выполнения программы дисплей остается в графическом режиме. Восстановление текстового режима возможно с помощью команды DOS MODE (MODE CO80) или пользовательской COM программой, в которой для этой цели используется команда INT 10H.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Память объемом 16К для цветного дисплея позволяет хранить дополнительные страницы (экраны). Возможны четыре страницы для экранов на 80 столбцов или восемь страниц для экранов на 40 столбцов.

- Графический режим обеспечивает низкое разрешение (не поддерживается в ROM), среднее разрешение (для цветной графики) и высокое разрешение (для черно-белой графики).

- Точка растра (минимальный элемент графического изображения) представляется определенным числом бит в зависимости от графического адаптера и разрешающей способности (низкой, средней или высокой).

- Для графики среднего разрешения на цветном графическом адаптере (CGA) можно выбрать четыре цвета, один из которых принадлежит к 16 возможным цветам, а три других формируют цветовую палитру.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

10.1. Сколько цветов возможно для фона и для текста на стандартном цветном адаптере (CGA) в текстовом режиме?

10.2. Напишите байты атрибуты в двоичном формате для а) сиреневого на ярко-голубом, б) коричневого на желтом, в)красного на сером с миганием.

10.3. Объясните разницу в количестве цветов, возможных при низком, среднем и высоком разрешении.

10.4. Напишите команды для вывода пяти символов карточной масти "бубны" в текстовом режиме ярко-зеленым цветом на сиреневом фоне.

10.5. Напишите команды для установки графического режима с разрешением а) 320x200 в адаптере CGA и б) 640x200 в адаптере EGA.

10.6. Напишите команды для установки синего фона в графическом режиме.

10.7. Напишите команды для чтения точки на 12 строке и 13 столбце в графическом режиме.

10.8. Модифицируйте программу на рис.10.1 для: а) графического режима на вашем мониторе; б) красного фона; в) строк с 10 по 30; г) столбцов с 20 по 300.

ГЛАВА 11 Команды обработки строк

Ц е л ь: Объяснить назначение специальных цепочечных команд, используемых для обработки символьных данных.

ВВЕДЕНИЕ

Команды, показанные в предыдущих главах, оперировали одним байтом, или одним словом за одно выполнение. Часто, однако, бывает необходимо переслать или сравнить поля данных, которые превышают по длине одно слово.

Например, необходимо сравнить описания или имена для того, чтобы отсортировать их в восходящей последовательности. Элементы такого формата известны как строковые данные и могут являться как символьными, так и числовыми. Для обработки строковых данных ассемблер имеет пять команд обработки строк:

MOVS - переслать один байт или одно слово из одной области памяти в другую;
LODS - загрузить из памяти один байт в регистр AL или одно слово в регистр AX;
STOS - записать содержимое регистра AL или AX в память;
CMPS - сравнить содержимое двух областей памяти, размером в один байт или в одно слово;
SCAS - сравнить содержимое регистра AL или AX с содержимым памяти.

Префикс REP позволяет этим командам обрабатывать строки любой длины.

СВОЙСТВА ОПЕРАЦИЙ НАД СТРОКАМИ

Цепочечная команда может быть закодирована для повторяющейся обработки одного байта или одного слова за одно выполнение. Например, можно выбрать "байтовую" команду для обработки строки с нечетным числом байт или "двухбайтовую" команду для обработки четного числа байт. Ниже перечислены регистры, участвующие в цепочечных командах (для однобайтовых и двухбайтовых вариантов). Предположим, что регистры DI и SI содержат необходимые адреса:

Команда	Операнды	Байт	Слово
MOVS	DI,SI	MOVSB	MOVSW
LODS	AL,SI или AX,SI	LODSB	LODSW
STOS	DI,AL или DI,AX	STOSB	STOSW
CMPS	SI,DI	CMPSB	CMPSW
SCAS	DI,AL или DI,AX	SCASB	SCASW

Например, можно кодировать операнды для команды MOVS, но опустить их для MOVSB и MOVSW. Эти команды предполагают, что регистры DI и SI содержат относительные адреса, указывающие на необходимые области памяти (для загрузки можно использовать команду LEA). Регистр SI обычно связан с регистром сегмента данных - DS:SI. Регистр DI всегда связан с регистром дополнительного сегмента - ES:DI. Следовательно, команды MOVS, STOS, CMPS и SCAS требуют инициализации регистра ES (обычно адресом в регистре DS).

REP: ПРЕФИКС ПОВТОРЕНИЯ ЦЕПОЧЕЧНОЙ КОМАНДЫ

Несмотря на то, что цепочечные команды имеют отношение к одному байту или одному слову, префикс REP обеспечивает повторение команды несколько раз. Префикс кодируется непосредственно перед цепочечной командой, например, REP MOVSB. Для использования префикса REP необходимо установить начальное значение в регистре CX. При выполнении цепочечной команды с префиксом REP происходит уменьшение на 1 значения в регистре CX до нуля. Таким образом, можно обрабатывать строки любой длины.

Флаг направления определяет направление повторяющейся операции:

- для направления слева направо необходимо с помощью команды CLD установить флаг DF в 0;
- для направления справа налево необходимо с помощью команды STD

установить флаг DF в 1.

В следующем примере выполняется пересылка 20 байт из STRING1 в STRING2. Предположим, что оба регистра DS и ES инициализированы адресом сегмента данных:

```
STRING1   DB   20 DUP('*')
STRING2   DB   20 DUP(' ')
          ...
          CLD              ;Сброс флага DF
          MOV  CX,20       ;Счетчик на 20 байт
          LEA  DI,STRING2  ;Адрес области "куда"
          LEA  SI,STRING1  ;Адрес области "откуда"
          REP MOVSB        ;Переслать данные
```

При выполнении команд CMPS и SCAS возможна установка флагов состояния, так чтобы операция могла прекратиться сразу после обнаружения необходимого условия. Ниже приведены модификации префикса REP для этих целей:

REP - повторять операцию, пока CX не равно 0;
REPZ или REPE - повторять операцию, пока флаг ZF показывает "равно или ноль". Прекратить операцию при флаге ZF, указывающему на не равно или не ноль или при CX равном 0;
REPNE или REPNZ - повторять операцию, пока флаг ZF показывает "не равно или не ноль". Прекратить операцию при флаге ZF, указывающему на "равно или нуль" или при CX равным 0.

Для процессоров 8086, 80286 и 80386, обрабатывающих слово за одно выполнение, использование цепочечных команд, где это возможно, приводит к повышению эффективности работы программы.

MOVS: ПЕРЕСЫЛКА СТРОК

На рис.7.5 была показана программа для пересылки девятибайтового поля. Программа включала три команды для инициализации и пять команд для цикла. Команда MOVS с префиксом REP и длиной в регистре CX может выполнять пересылку любого числа символов более эффективно.

Для области, принимающей строку, сегментным регистром, является регистр ES, а регистр DI содержит относительный адрес области, передающей строку. Сегментным регистром является регистр DS, а регистр SI содержит относительный адрес. Таким образом, в начале программы перед выполнением команды MOVS необходимо инициализировать регистр ES вместе с регистром DS, а также загрузить требуемые относительные адреса полей в регистры DI и SI. В зависимости от состояния флага DF команда MOV S производит увеличение или уменьшение на 1 (для байта) или на 2 (для слова) содержимого регистров DI и SI.

Приведем команды, эквивалентные цепочечной команде REP MOVSB:

```
         JCXZ     LABEL2
LABEL1:  MOV      AL,[SI]
         MOV      [DI],AL
         INC/DEC  DI       ;Инкремент или декремент
         UNC/DEC  SI       ;Инкремент или декремент
         LOOP     LABEL1
LABEL2:  ...
```

В программе на рис.11.1 процедура C10MVSB использует команду MOVSB для пересылки содержимого десятибайтового поля NAME1 в поле NAME2. Первая

команда CLD сбрасывает флаг направления в 0 для обеспечения процесса пересылки слева направо. В нормальном состоянии флаг DF обычно имеет нулевое значение и команда CLD используется из предосторожности.

```
      page    60,132
TITLE   STRING  (EXE)  Проверка строковых операций
; --------------------------------------------------
STACKSG SEGMENT PARA STACK 'Stack'
      DW      32 DUP(?)
STACKG  ENDS
; --------------------------------------------------
DATASG  SEGMENT PARA 'Data'
NAME1   DB      'Assemblers'    ;Элементы данных
NAME2   DB      10 DUP(' ')
NAME3   DB      10 DUP(' ')
DATASG  ENDS
; --------------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR             ;Основная процедура
      ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASG
      PUSH    DS
      SUB     AX,AX
      PUSH    AX
      MOV     AX,DATASG
      MOV     DS,AX
      MOV     ES,AX
      CALL    C10MVSB         ;Подпрограмма MVSB
      CALL    D10MVSW          ;Подпрограмма LODS
      CALL    E10LODS         ;Подпрограмма LODS
      CALL    F10STOS         ;Подпрограмма CMPS
      CALL    H10SCAS         ;Подпрограмма SCAS
      RET
BEGIN   ENDP
;             Использование MOVSB:
;             -------------------
C10MVSB PROC    NEAR
      CLD
      LEA     SI,NAME1
      LEA     D1,NAME2
      MOV     CX,10           ;Переслать 10 байтов
      REP MOVSB                ; из NAME1 в NAME2
      RET
C10MVSB ENDP
;             Использование MOVSW:
;             -------------------
D10MVSW PROC    NEAR
      CLD
      LEA     SI,NAME2
      LEA     DI,NAME3
      MOV     CX,05           ;Переслать 5 слов
      REP MOVSW                 ; из NAME2 в NAME3
      RET
D10MVSW ENDP
;             Использование LODSW:
;             -------------------
E10LODS PROC    NEAR
      CLD
      LEA     SI,NAME1         ;Загрузить первое слово
      LODSW                  ; из NAME1 в AX
```

```
       RET
E10LODS ENDP
;             Использование STOSW:
;             -------------------
F10STOS PROC    NEAR
       CLD
       LEA     D1,NAME3
       MOV     CX,05
       MOV     AX,2020H        ;Переслать пробелы
       REP STOSW               ; в NAME3
       RET
F10STOS ENDP
;             Использование CMPSB:
;             -------------------
G10CMPS PROC    NEAR
       CLD
       MOV     CX,10
       LEA     SI,NAME1
       LEA     DI,NAME2
       REPE CMPSB              ;Сравнить NAME1 и NAME2
       JNE     G20             ;Не равны?
       MOV     BH,01

G20:   MOV     CX,10
       LEA     SI,NAME2
       LEA     DI,NAME3
       REPE CMPSB              ;Сравнить NAME2 и NAME3
       JE      G30             ;Если равны, то выйти
       MOV     BL,02
G30:   RET
G10CMPS ENDP

;             Использование SCASB:
;             -------------------
H10SCAS PROC    NEAR
       CLD
       MOV     CX,10
       LEA     DI,NAME1
       MOV     AL,'m'          ;Поиск символа 'm'
       REPNE SCASB             ; в NAME1
       JNE     H20             ;Если не найден - выйти
       MOV     AH,03
H20:   RET
H10SCAS ENDP

CODES   ENDS
       END     BEGIN
```

Рис.11.1. Использование цепочечных команд.

Две команды LEA загружают регистры SI и DI относительными адресами NAME1 и NAME2 соответственно. Так как регистры DS и ES были ранее инициализированы адресом DATASG, то полные адреса полей NAME1 и NAME2 будут в регистрах ES:DI и DS:SI. (COM программа автоматически инициализирует регистры ES и DS). Команда MOV заносит в регистр CX значение 10 - длину полей NAME1 и NAME2. Команда REP MOVSB выполняет следующее:

- Пересылает самый левый байт из поля NAME1 (адресованного

регистрами ES:DI) в самый левый байт поля NAME2 (адресованного регистрами DS:SI).

- Увеличивает на 1 адреса в регистрах DI и SI для следующего байта.
- Уменьшает CX на 1.
- Повторяет перечисленные действия (в данном случае 10 раз), пока содержимое регистра CX не станет равным нулю.

Поскольку флаг DF имеет нулевое значение, команда MOVSB увеличивает адреса в регистрах DI и SI, и в каждой итерации процесс переходит на байт вправо, т.е. пересылает байт из NAME1+1 в NAME2+1 и т.д. Если бы флаг DF был равен 1, тогда команда MOVSB уменьшала бы адреса в регистрах DI и SI, выполняя процесс справа налево. Но в этом случае регистры SI и DI необходимо инициализировать адресами последних байтов полей, т.е. NAME1+9 и NAME2+9 соответственно.

В процедуре D10MVSW (рис.11.1) используется команда MOVSW, пересылающая одно слово за одно выполнение. Так как команда MOVSW увеличивает адреса в регистрах DS и SI на 2, операция требует только пять циклов. Для процесса пересылки справа налево регистр SI должен быть инициализирован адресом NAME1+8, а регистр DI - NAME2+8.

LODS: ЗАГРУЗКА СТРОКИ

Команда LODS загружает из памяти в регистр AL один байт или в регистр AX одно слово. Адрес памяти определяется регистрами DS:SI. В зависимости от значения флага DF происходит увеличение или уменьшение регистра SI.

Поскольку одна команда LODS загружает регистр, то практической пользы от префикса REP в данном случае нет. Часто простая команда MOV полностью адекватна команде LODS, хотя MOV генерирует три байта машинного кода, а LODS - только один, но требует инициализацию регистра SI. Можно использовать команду LODS в том случае, когда требуется продвигаться вдоль строки (по байту или по слову), проверяя загружаемый регистр на конкретное значение.

Команды, эквивалентные команде LODSB:

```
MOV  AL,[SI]
INC  SI
```

На рис.11.1 процедура E10LODS демонстрирует использование команды LODSW. В примере обрабатывается только одно слово: первый байт из области NAME1 (содержащий As) заносится в регистр AL, а второй байт - в регистр AH. В результате в регистре AX получится значение sA.

STOS: ЗАПИСЬ СТРОКИ

Команда STOS записывает (сохраняет) содержимое регистра AL или AX в байте или в слове памяти. Адрес памяти всегда представляется регистрами ES:DI. В зависимости от флага DF команда STOS также увеличивает или уменьшает адрес в регистре DI на 1 для байта или на 2 для слова.

Практическая польза команды STOS с префиксом REP - инициализация области данных конкретным значением, например, очистка дисплейного буфера пробелами. Длина области (в байтах или в словах) загружается в регистр AX. Команды, эквивалентные команде REP STOSB:

```
        JCXZ     LABEL2
LABEL1: MOV      [DI],AL
        INC/DEC  DI       ;Инкремент или декремент
```

```
          LOOP      LABEL1
     LABEL2:   ...
```

На рис.11.1 процедура F10STOS демонстрирует использование команды STOSW. Операция осуществляет запись шест. 2020 (пробелы) пять раз в область NAME3, причем значение из регистра AL заносится в первый байт, а из регистра AH - во второй. По завершении команды регистр DI содержит адрес NAME3+10.

CMPS: СРАВНЕНИЕ СТРОК

Команда CMPS сравнивает содержимое одной области памяти (адресуемой регистрами DS:SI) с содержимыми другой области (адресуемой как ES:DI). В зависимости от флага DF команда CMPS также увеличивает или уменьшает адреса в регистрах SI и DI на 1 для байта или на 2 для слова. Команда CMPS устанавливает флаги AF, CF, OF, PF, SF и ZF. При использовании префикса REP в регистре CX должна находиться длина сравниваемых полей. Команда CMPS может сравнивать любое число байт или слов.

Рассмотрим процесс сравнения двух строк, содержащих имена JEAN и JOAN. Сравнение побайтно слева направо приводит к следующему:

```
          J  :  J       Равно
          E  :  O       Не равно (E меньше O)
          A  :  A       Равно
          N  :  N       Равно
```

Сравнение всех четырех байт заканчивается сравнением N:N - равно/нуль. Так как имена "не равны", операция должна прекратиться, как только будет обнаружено условие "не равно". Для этих целей команда REP имеет модификацию REPE, которая повторяет сравнение до тех пор, пока сравниваемые элементы равны, или регистр CX не равен нулю. Кодируется повторяющееся однобайтовое сравнение следующим образом:

```
          REPE CMPSB
```

На рис.11.1 в процедуре G10CMPS имеются два примера использования команды CMPSB. В первом примере происходит сравнение содержимого полей NAME1 и NAME2. Так как ранее команда MOVSB переслала содержимое поля NAME1 в поле NAME2, то команда CMPSB продолжается на всех десяти байтах и завершается состоянием равно/нуль: флаг SF получает значение 0 (положительно) и флаг ZF - 1(нуль).

Во втором примере сравнивается поля NAME2 и NAME3. Ранее команда STOSW заполнила поле NAME3 пробелами, поэтому команда CMPB завершается после сравнения первых же байт с результатом "больше/неравно": флаг SF получает значение 0 (положительно) и флаг ZF - 0 (не нуль).

Первый пример заканчивается с результатом "равно/нуль" и заносит 01 в регистр BH. Второй пример заканчивается с результатом "неравно" и заносит 02 в регистр BL. При трассировке команд с помощью отладчика DEBUG можно увидеть, что в конце процедуры G10CMPS регистр BX будет содержать значение 0102.

Предупреждение! Показанные примеры используют команду CMPSB для сравнения одного байта за одно выполнение. При использовании команды CMPSW для сравнения одного слова, необходимо инициализировать регистр CX значением 5. Кроме того следует помнить, что команда CMPSW при сравнении слов переставляет байты. Например, сравнивая имена SAMUEL и ARNOLD команда CMPSW выбирает вместо SA и AR переставленные значения, т.е. AS и RA. В результате вместо "больше" получится "меньше", т.е. неправильный результат. Таким образом команда CMPSW работает правильно только при сравнении строк, которые содержат числовые данные, определенные как DW, DD

или DQ.

SCAS: СКАНИРОВАНИЕ СТРОК

Команда SCAS отличается от команды CMPS тем, что сканирует (просматривает) строку на определенное значение байта или слова. Команда SCAS сравнивает содержимое области памяти (адресуемой регистрами ES:DI) с содержимым регистра AL или AX. В зависимости от значения флага DF команда SCAS также увеличивает или уменьшает адрес в регистре DI на 1 для байта или на 2 для слова. Команда SCAS устанавливает флаги AF, CF, OF, PF, SF и ZF. При использовании префикса REP и значения длины в регистре CX команда SCAS может сканировать строки любой длины.

Команда SCAS особенно полезна, например, в текстовых редакторах, где программа должна сканировать строки, выполняя поиск знаков пунктуации: точек, запятых и пробелов.

На рис.11.1 процедура H10SCAS сканирует область NAME1 на строчную букву "m". Так как команда SCASB должна продолжать сканирование, пока результат сравнения - "не равно" или регистр CX не равен нулю, то используется префикс REPNE:

```
        REPNE SCASB
```

Так как область NAME1 содержит слово "Assemblers", то команда SCASB находит символ "m" в пятом сравнении. При использовании отладчика DEBUG для трассировки команд в конце процедуры H10SCAS можно увидеть в регистре AH значение 03 для индикации того, что символ "m" найден. Команда REP SCASB кроме того уменьшит значение регистра CX от 10 до 06.

Команда SCASW сканирует в памяти слово на соответствие значению в регистре AX. При использовании команд LODSW или MOV для пересылки слова в регистр AX, следует помнить, что первый байт будет в регистре AL, а второй байт - в регистре AH. Так как команда SCAS сравнивает байты в обратной последовательности, то операция корректна.

СКАНИРОВАНИЕ И ЗАМЕНА

В процессе обработки текстовой информации может возникнуть необходимость замены определенных символов в тексте на другие, например, подстановка пробелов вместо различных редактирующих символов. В приведенном ниже фрагменте программы осуществляется сканирование строки STRING и замена символа амперсанд (&) на символ пробела. Когда команда SCASB обнаружит символ & (в примере это будет позиция STRING+8), то операция сканирования прекратится и регистр DI будет содержать адрес STRING+9. Для получения адреса символа & необходимо уменьшить содержимое DI на единицу и записать по полученному адресу символ пробела.

```
        STRLEN    EQU  15            ;Длина поля STRING
        STRING    DB   'The time&is now'
                ...
                CLD
                MOV  AL,'&'          ;Искомый символ
                MOV  CX,STRLEN       ;Длина поля STRING
                LEA  DI,STRING       ;Адрес поля STRING
                REPNE SCASB          ;Сканировать
                JNZ  K20             ;Символ найден?
                DEC  DI              ;Да - уменьшить адрес
                MOV  BYTE PTR[DI],20H ;Подставить пробел
        K20:      RET
```

АЛЬТЕРНАТИВНОЕ КОДИРОВАНИЕ

При использовании команд MOVSB или MOVSW ассемблер предполагает наличие корректной длины строковых данных и не требует кодирования операндов в команде. Для команды MOVS длина должна быть закодирована в операндах. Например, если поля FLDA и FLDB определены как байтовые (DB), то команда

```
REP  MOVS FLDA,FLDB
```

предполагает повторяющуюся пересылку байтов из поля FLDB в поле FLDA. Эту команду можно записать также в следующем виде:

```
REP  MOVS ES:BYTE PTR[DI],DS:[SI]
```

Однако загрузка регистров DI и SI адресами FLDA и FLDB обязательна в любом случае.

ДУБЛИРОВАНИЕ ОБРАЗЦА

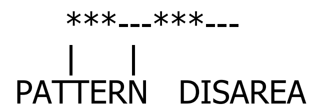
Команда STOS бывает полезна для установки в некоторой области определенных значений байтов и слов. Для дублирования образца, длина которого превышает размер слова, можно использовать команду MOVS с небольшой модификацией. Предположим, что необходимо сформировать строку следующего вида:

```
***---***---***---***---***--- . . .
```

Вместо того, чтобы определять полностью всю строку, можно определить только первые шесть байтов. Закодируем образец непосредственно перед обрабатываемой строкой следующим образом:

```
PATTERN   DB   '***---'
DISAREA   DB   42 DUP(?)
          .
          .
          CLD
          MOV  CX,21
          LEA  DI,DISAREA
          LEA  SI,PATTERN
          REP MOVSW
```

В процессе выполнения команда MOVSW сначала пересылает первое слово (**) из образца PATTERN в первое слово области DISAREA, затем - второе слово (*-), потом третье (--):



К этому моменту регистр DI будет содержать адрес DISAREA+6, а регистр SI - PATTERN+6, который также является адресом DISAREA. Затем команда MOVSW автоматически дублирует образец, пересылая первое слово из DISAREA в DISAREA+6, из DISAREA+2, в DISAREA+8, из DISAREA+4 в DISAREA+10 и т.д. В результате образец будет полностью продублирован по всей области DISAREA:

```
***---***---***---***---***--- . . . ***---
```

```
          |         |          |          |
      PATTERN     DISAREA+6   DISAREA+12   DISAREA+42
```

Данную технику можно использовать для дублирования в области памяти любого образца любой длины. Образец должен быть расположен непосредственно перед принимающей областью.

ПРОГРАММА: ВЫРАВНИВАНИЕ ВПРАВО ПРИ ВЫВОДЕ НА ЭКРАН

COM-программа, изображенная на рис.11.2, иллюстрирует почти весь материал, приведенный в этой главе. Процедуры программы выполняют следующие действия:

B10INPT - Принимает имена длиной до 30 символов, вводимых вверху экрана.
D10SCAS - Использует команду SCASB для сканирования имен и об хода любого ввода, содержащего символ "звездочка".
E10RGHT - Использует команду MOVSB для выравнивания имен по правой границе выводит имена в колонку в правой части экрана. Длина в поле ACTNLEN из списка параметров ввода используется для вычисления самого правого символа в имени, например:

```
                JEROME KERN
            OSCAR HAMMERSTEIN
              RICHARD ROGERS
```

F10CLNM - Использует команду STOSW для очистки области имени в памяти.

```
      page    60,132
TITLE   EXRING  (COM) Вывод имен, выровненных справа
CODESG  SEGMENT PARA 'Code'
      ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:CODESG
      ORG     100H
BEGIN:  JMP     SHORT MAIN
;-----------------------------------------------------
NAMEPAR LABEL   BYTE            ;Имя списка параметров
MAXNLEN DB      31             ;Макс. длина
ACTNLEN DB      ?              ;Число введенных символов
NAMEFLD DB      31 DUP(' ')     ;Имя

PROMPT  DB      'Name?', '$'
NAMEDSP DB      31 DUP(' '), 13, 10, '$'
ROW     DB      00
;-----------------------------------------------------
MAIN    PROC    NEAR            ;Основная процедура
      MOV     AX,0600H
      CALL    Q10SCR          ;Очистить экран
      SUB     DX,DX           ;Установить курсор в 00,00
      CALL    Q20CURS
A10LOOP:
      CALL    B10INPT         ;Ввести имя с клавиатуры
      TEST    ACTNLEN,0FFH    ;Нет имени? (т.е. конец)
      JZ      A90             ; да - выйти
      CALL    D10SCAS         ;Найти звездочку
      CMP     AL,'*'          ;Найдена?
      JE      A10LOOP         ; да - обойти
      CALL    E10RGHT         ;Выровнять имя справа
      CALL    A10LOOP
```

```
A90:    RET
MAIN    ENDP
;              Вывод запроса для ввода имени:
;              ----------------------------
B10INPT PROC
       MOV     AH,09
       LEA     DX,PROMPT       ;Выдать текст запроса
       INT     21H
       RET
B10INPT ENDP
;              Поиск звездочки в имени:
;              ----------------------
D10SCAS PROC
       CLD
       MOV     AL,'*'
       MOV     CX,30           ;Длина сканирования - 30
       LEA     DI,NAMEFLD
       REPNE SCASB             ;Звездочка найдена?
       JE      D20             ; да - выйти,
       MOV     AL,20H          ; нет стереть * в AL
D20:   RET
D10SCAS ENDP
;              Выравнивание справа и вывод на экран:
;              -----------------------------------
E10RGHT PROC
       STD
       SUB     CH,CH
       MOV     CL,ACTNLEN      ;Длина в CX для REP
       LEA     SI,NAMEFLD      ;Вычислить самую правую
       ADD     SI,CX           ; позицию
       DEC     SI              ; введенного имени
       LEA     DI,NAMEDSP+30   ;Правая поз. поля имени
       REP MOVSB               ;Переслать справа налево
       MOV     DH,ROW
       MOV     DL,48
       CALL    Q20CURS         ;Установить курсор
       MOV     AH,09
       LEA     DX,NAMEDSP      ;Выдать имя на экран
       INT     21H
       CMP     ROW,20          ;Последняя строка экрана?
       JAE     E20             ; нет -
       INC     ROW             ; увеличить строку,
       JMP     E90
E20:
       MOV     AX,0601H        ; да -
       CALL    Q10SCR          ; прокрутить и
       MOV     DH,ROW          ; установить курсор
       MOV     DL,00
       CALL    Q20CURS
E90:   RET
E10RGHT ENDP
;              Очистить область имени:
;              ---------------------
F10CLNM PROC
       CLD
       MOV     AX,2020H
       MOV     CX,15           ;Очистить 15 слов
       LEA     DI,NAMEDSP
       REP STOSW
       RET
F10CLNM ENDP
```

```
;              Прокрутка экрана:
;              ----------------
Q10SCR  PROC                  ;AX установлен при вызове
      MOV     BH,30          ;Цвет ( 07 для ч/б)
      MOV     CX,00
      MOV     DX,184FH
      INT     10H
      RET
Q10SCR  ENDP
;              Установить курсор (строка/столбец):
;              ---------------------------------
Q20CURS PROC                  ;DX установлен при вызове
      MOV     AH,02
      SUB     BH,BH
      INT     10H
      RET
Q20CURS ENDP
CODESG  ENDS
      END     BEGIN
```

Рис.11.2. Выравнивание вправо при выводе на экран.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Для цепочечных команд MOVS, STOS, CMPS и SCAS не забывайте инициализировать регистр ES.

- Сбрасывайте (CLD) или устанавливайте (STD) флаг направления в соответствии с направлением обработки.

- Не забывайте устанавливать в регистрах DI и SI необходимые значения. Например, команда MOVS предполагает операнды DI,SI, а команда CMPS - SI,DI.

- Инициализируйте регистр CX в соответствии с количеством байтов или слов, участвующих в процессе обработки.

- Для обычной обработки используйте префикс REP для команд MOVS и STOS и модифицированный префикс (REPE или REPNE) для команд CMPS и SCAS.

- Помните об обратной последовательности байтов в сравниваемых словах при выполнении команд CMPSW и SCASW.

- При обработке справа налево устанавливайте начальные адреса на последний байт обрабатываемой области. Если, например, поле NAME1 имеет длину 10 байтов, то для побайтовой обработки данных в этой области справа налево начальный адрес, загружаемый командой LEA, должен быть NAME1+9. Для обработки слов начальный адрес в этом случае - NAME1+8.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

11.1. В данной главе приведены эквивалентные команды для а) MOVSB, б) LODSB и в) STOSB с префиксом REP. Напишите эквивалентные команды для обработки по словам а) MOVSW, б) LODSW и в) STOSW с префиксом REP.

11.2. Введите, ассемблируйте и выполните компоновку программы,

приведенной на рис.11.1. Не забудьте о инициализации регистра ES. Замените команды MOVSB и MOVSW для пересылки справа налево. Измените процедуру H10SCAS для сканирования поля NAME1 на слово "mb". Используя отладчик DEBUG для трассировки процедур, обратите внимание на содержимое сегмента данных и регистров.

11.3. Имеются следующие определения:

```
DATASG   SEGMENT   PARA
         CONAME    DB   'SPACE EXPLORERS INC.'
         PRLINE    DB   20 DUP(' ')
```

Используя цепочечные команды, выполните:

а) пересылку данных из CONAME в PRLINE слева направо;
б) пересылку данных из CONAME в PRLINE справа налево;
в) загрузку третьего и четвертого байтов области CONAME в регистр AX;
г) сохранение содержимого регистра AX в область по адресу PRLINE+5;
д) сравнение данных в областях CONAME и PRLINE (они должны быть не равны);
е) сканирование областей CONAME и PRLINE, и поиск в ней символа пробел. Если символ будет найден, то переслать его в регистр BH.

11.4. Переделайте процедуру H10SCAS (рис.11.1) так, чтобы выполнялось сканирование поля NAME1 на символ "er". Обратите внимание, что символы "er" не встречаются в поле NAME1 как одно слово: /As/se/mb/le/rs/. Для решения этой проблемы возможны два варианта:

а) использовать команду SCASW дважды, причем первая должна начинаться по адресу NAME1, а вторая - по адресу NAME1+1;
б) использовать команду SCASB для поиска символа "e" и сравнить затем следующий байт на символ "r".

11.5. Определите поле, содержащее шест. значения 03, 04, 05 и B4. Продублируйте это поле 20 раз и выдайте результат на экран.

ГЛАВА 12 Арифметические операции I: Обработка двоичных данных

Ц е л ь: Дать сведения об операциях сложения, вычитания, умножения и деления двоичных данных.

ВВЕДЕНИЕ

Несмотря на то, что мы привыкли к десятичной арифметике (база 10), компьютер работает только с двоичной арифметикой (база 2). Кроме того, ввиду ограничения, накладываемого 16-битовыми регистрами, большие величины требуют специальной обработки.

Данная глава дает сведения об операциях сложения, вычитания, умножения и деления для беззнаковых и знаковых данных. В главе приводятся много примеров и предупреждений о различных ловушках для опрометчивых исследователей мира микропроцессора. В следующей главе будут раскрыты операции преобразования между двоичными данными и ASCII кодами.

СЛОЖЕНИЕ И ВЫЧИТАНИЕ

Команды ADD и SUB выполняют сложение и вычитание байтов или слов, содержащих двоичные данные. Вычитание выполняется в компьютере по методу сложения с двоичным дополнением: для второго операнда устанавливаются обратные значения бит и прибавляется 1, а затем происходит сложение с первым операндом. Во всем, кроме первого шага, операции сложения и вычитания идентичны.

На рис.12.1 представлены примеры команд ADD и SUB, обрабатывающие байты или слова. В процедуре B10ADD используется команда ADD для сложения байтов, а в процедуре C10SUB команда SUB вычитает слова. Примеры показывают все пять возможных ситуаций:

сложение/вычитание регистр-регистр;
сложение/вычитание память-регистр;
сложение/вычитание регистр-память;
сложение/вычитание регистр-непоср.значение;
сложение/вычитание память-непоср.значение.

```
       page    60,132
TITLE   EXADD   (COM) Сложение и вычитание
CODESG  SEGMENT PARA 'Code'
       ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
       ORG     100H
BEGIN:  JMP     SHORT MAIN
; -------------------------------------------
BYTEA   DB      64H              ;Элементы данных
BYTEB   DB      40H
BYTEC   DB      16H
WORDA   DW      4000H
WORDB   DW      2000H
WORDC   DW      1000H
; -------------------------------------------
MAIN    PROC    NEAR             ;Основная процедура:
       CALL    B10ADD           ;Вызвать сложение ADD
       CALL    C10SUB           ;Вызвать вычитание SUB
       RET
MAIN    ENDP
;              Пример сложения байт:
;              --------------------
B10ADD  PROC
       MOV     AL,BYTEA
       MOV     BL,BYTEB
       ADD     AL,BL            ;Регистр и регистр
       ADD     AL,BYTEC         ;Память и регистр
       ADD     BYTEA,BL         ;Регистр и память
       ADD     BL,10H           ;Непосредств. и регистр
       ADD     BYTEA,25H        ;Непосредств. и память
       RET
B10ADD  ENDP
;              Пример вычитания слов:
;              ---------------------
C10SUB  PROC
       MOV     AX,WORDA
       MOV     BX,WORDB
       SUB     AX,BX            ;Регистр из регистра
       SUB     AX,WORDC         ;Память из регистра
       SUB     WORDA,BX         ;Регистр из памяти
       SUB     BX,1000H         ;Непосредств. из рег.
       SUB     WORDA,256H       ;Непосредств. из пам.
```

```
      RET
C10SUB  ENDP
CODESG  ENDS
      END     BEGIN
```

Рис.12.1. Примеры команд ADD и SUB.

Поскольку прямой операции память-память не существует, данная операция выполняется через регистр. В следующем примере к содержимому слова WORDB прибавляется содержимое слова WORDA, описанных как DW:

```
MOV  AX,WORDA
ADD  AX,WORDB
MOV  WORDB,AX
```

Переполнения

Опасайтесь переполнений в арифметических операциях. Один байт содержит знаковый бит и семь бит данных, т.е. значения от -128 до +127. Результат арифметической операции может легко превзойти емкость однобайтового регистра. Например, результат сложения в регистре AL, превышающий его емкость, автоматически не переходит в регистр AH. Предположим, что регистр AL содержит шест.60, тогда результат команды

```
ADD  AL,20H
```

генерирует в AL сумму - шест.80. Но операция также устанавливает флаг переполнения и знаковый флаг в состояние "отрицательно". Причина заключается в том, что шест.80 или двоичное 1000 0000 является отрицательным числом. Т.е. в результате, вместо +128, мы получим -128. Так как регистр AL слишком мал для такой операции и следует воспользоваться регистром AX. В следующем примере команда CBW (Convert Byte to Word - преобразовать байт в слово) преобразует шест.60 в регистре AL в шест.0060 в регистре AX, передавая при этом знаковый бит (0) через регистр AH. Команда ADD генерирует теперь в регистре AX правильный результат: шест.0080, или +128:

```
CBW           ;Расширение AL до AX
ADD  AX,20H    ;Прибавить к AX
```

Но полное слово имеет также ограничение: один знаковый бит и 15 бит данных, что соответствует значениям от -32768 до +32767. Рассмотрим далее как можно обрабатывать числа, превышающие эти пределы.

Многословное сложение

Максимальное возможное значение в регистре +32767 ограничивает возможность компьютера для выполнения арифметических операций. Рассмотрим два способа выполнения арифметических операций. Первый способ - более прост, но специфичен, второй - сложнее, но имеет общий характер.

```
      page    60,132
TITLE   EXDBADD (COM) Пример сложения двойных слов
CODESG  SEGMENT PARA 'Code'
      ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
```

```
       ORG     100H
BEGIN:  JMP     SHORT MAIN
; ------------------------------------------
WORD1A  DW      0123H            ;Элементы данных
WORD1B  DW      0BC62H
WORD2A  DW      0012H
WORD2B  DW      553AH
WORD3A  DW      ?
WORD3B  DW      ?
; ------------------------------------------
MAIN    PROC    NEAR              ;0сновная процедура:
       CALL    D10DWD           ;Вызвать сложение 1
       CALL    E10DWD           ;Вызвать сложение 2
       RET
MAIN    ENDP
;             Пример сложения двойных слов:
;             ----------------------------
D10DWD  PROC
       MOV     AX,WORD1B         ;Сложить правые слова
       ADD     AX,WORD2B
       MOV     WORD3B,AX
       MOV     AX,WORD1A         ;Сложить левые слова
       ADC     AX,WORD2A         ; с переносом
       MOV     WORD3A,AX
       RET
D10DWD  ENDP
;            Сложение чисел любой длины:
;            -------------------------
E10DWD  PROC
       CLC                     ;Очистить флаг переноса
       MOV     CX,2              ;Установить счетчик
       LEA     SI,WORD1B          ;Левое слово DWORD1
       LEA     DI,WORD2B          ;Левое слово DWORD2
       LEA     BX,WORD3B          ;Левое слово суммы
E20:
       MOV     AX,[SI]           ;Поместить слово в AX
       ADC     AX,[DI]           ;Сложить с переносом
       MOV     [BX],AX           ;Сохранить слово
       DEC     SI
       DEC     SI
       DEC     DI
       DEC     DI
       DEC     BX
       DEC     BX
       LOOP    E20               ;Повторить цикл
       RET
E10DWD  ENDP
CODESG  ENDS
       END     BEGIN
```

Рис.12.2. Сложение двойных слов.

На рис.12.2 процедура D10DWD демонстрирует простой способ сложения содержимого одной пары слов (WORD1A и WORD1B) с содержимым второй пары слов (WORD2A и WORD2B) и сохранения суммы в третьей паре слов (WORD3A и WORD3B). Сначала выполняется сложение правых слов:

```
WORD1B      BC62
WORD2B      553A
```

```
          -----
     Сумма:        1119C
```

Сумма - шест.1119C превышает емкость регистра AX. Переполнение вызывает установку флага переноса в 1. Затем выполняется сложение левых слов, но в данном случае, вместо команды ADD используется команда сложения с переносом ADC (ADd with Carry). Эта команда складывает два значения, и если флаг CF уже установлен, то к сумме прибавляется 1:

```
     WORD1A        0123
     WORD2A        0012
     Плюс перенос     1
          ----
     Сумма:        0136
```

При использовании отладчика DEBUG для трассировки арифметических команд можно увидеть эту сумму 0136 в регистре AX, и обратные значения 3601 в поле WORD3A и 9C11 в поле WORD3B.

На рис.12.2 процедура E10DWD демонстрирует подход к сложению значений любой длины. Действие начинается со сложения самых правых слов складываемых полей. В первом цикле складываются правые слова, во втором - слова, расположенные левее. При этом адреса в регистрах SI, DI и BX уменьшаются на 2. По две команда DEC выполняют эту операцию для каждого регистра. Применять команду

```
     SUB  reg,02
```

в данном случае нельзя, т.к. при этом будет очищен флаг переноса, что приведет к искажению результата сложения.

Ввиду наличия цикла, используется только одна команда сложения ADC. Перед циклом команда CLC (CLear Carry - очистить флаг переноса) устанавливает нулевое значение флага переноса. Для работы данного метода необходимо: 1) обеспечить смежность слов, 2) выполнять обработку справа налево и 3) загрузить в регистр CX число складываемых слов.

Для многословного вычитания используется команда SBB (SuBtract with Borrow - вычитание с заемом) эквивалентная команде ADC. Заменив в процедуре E10DWD (рис.12.2) команду ADC на SBB, получим процедуру для вычитания.

БЕЗЗНАКОВЫЕ И ЗНАКОВЫЕ ДАННЫЕ

Многие числовые поля не имеют знака, например, номер абонента, адрес памяти. Некоторые числовые поля предлагаются всегда положительные, например, норма выплаты, день недели, значение числа ПИ. Другие числовые поля являются знаковые, так как их содержимое может быть положительным или отрицательным. Например, долговой баланс покупателя, который может быть отрицательным при переплатах, или алгебраическое число.

Для беззнаковых величин все биты являются битами данных и вместо ограничения +32767 регистр может содержать числа до +65535. Для знаковых величин левый байт является знаковым битом. Команды ADD и SUB не делают разницы между знаковыми и беззнаковыми величинами, они просто складывают и вычитают биты. В следующем примере сложения двух двоичных чисел, первое число содержит единичный левый бит. Для беззнакового числа биты представляют положительное число 249, для знакового - отрицательное число -7:

```
               Беззнаковое   Знаковое
     11111001       249          -7
     00000010         2          +2
```

```
               ---         --
      11111011         251         -5
```

Двоичное представление результата сложения одинаково для беззнакового и знакового числа. Однако, биты представляют +251 для беззнакового числа и -5 для знакового. Таким образом, числовое содержимое поля может интерпретироваться по разному.

Состояние "перенос" возникает в том случае, когда имеется перенос в знаковый разряд. Состояние "переполнение" возникает в том случае, когда перенос в знаковый разряд не создает переноса из разрядной сетки или перенос из разрядной сетки происходит без переноса в знаковый разряд. При возникновении переноса при сложении беззнаковых чисел, результат получается неправильный:

```
             Беззнаковое   Знаковое  CF   OF
      11111100         252          -4
      00000101           5         +5
               ---         --
      00000001           1          1    1    0
             (неправильно)
```

При возникновении переполнения при сложении знаковых чисел, результат получается неправильный:

```
             Беззнаковое   Знаковое  CF   OF
      01111001         121        +121
      00001011          11        +11
               ---        ----
      10000100         132        -124    0    1
                    (неправильно)
```

При операциях сложения и вычитания может одновременно возникнуть и переполнение, и перенос:

```
             Беззнаковое   Знаковое  CF   OF
      11110110         246         -10
      10001001         137        -119
               ---        ----
      01111111         127        +127    1    1
             (неправильно)  (неправильно)
```

УМНОЖЕНИЕ

Операция умножения для беззнаковых данных выполняется командой MUL, а для знаковых - IMUL (Integer MULtiplication - умножение целых чисел). Ответственность за контроль над форматом обрабатываемых чисел и за выбор подходящей команды умножения лежит на самом программисте. Существуют две основные операции умножения:

"Б а й т н а б а й т". Множимое находится в регистре AL, а множитель в байте памяти или в однобайтовом регистре. После умножения произведение находится в регистре AX. Операция игнорирует и стирает любые данные, которые находились в регистре AH.

```
               | AH |   AL   |            |    AX      |
   До умножения: |    |Множимое|     После: |Произведение|
```

"С л о в о н а с л о в о". Множимое находится в регистре AX, а множитель - в слове памяти или в регистре. После умножения произведение

находится в двойном слове, для которого требуется два регистра: старшая (левая) часть произведения находится в регистре DX, а младшая (правая) часть в регистре AX. Операция игнорирует и стирает любые данные, которые находились в регистре DX.

```
                 |  AX  |          |  DX  ||  AX  |
  До умножения:|Множимое|  После: |Ст.часть||Мл.часть|
                                   |  Произведение  |
```

В единственном операнде команд MUL и IMUL указывается множитель. Рассмотрим следующую команду:

```
          MUL  MULTR
```

Если поле MULTR определено как байт (DB), то операция предполагает умножение содержимого AL на значение байта из поля MULTR. Если поле MULTR определено как слово (DW), то операция предполагает умножение содержимого AX на значение слова из поля MULTR. Если множитель находится в регистре, то длина регистра определяет тип операции, как это показано ниже:

```
       MUL  CL  ;Байт-множитель: множимое в AL, произвед. в AX
       MUL  BX  ;Слово-множитель:множимое в AX, произвед. в DX:AX
```

Беззнаковое умножение: Команда MUL

Команда MUL (MULtiplication - умножение) умножает беззнаковые числа. На рис.12.3 в процедуре C10MUL дано три примера умножения: байт на байт, слово на слово и слово на байт. Первый пример команды MUL умножает шест.80 (128) на шест.47 (64). Произведение шест.2000 (8192) получается в регистре AX.

```
      page 60,132
TITLE   EXMULT  (COM) Пример команд умножения
CODESG  SEGMENT PARA 'Code'
      ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
      OR6     100H
BEGIN:  JMP     SHORT MAIN
; -------------------------------------------
BYTE1   DB      80H
BYTE2   DB      40H
WORD1   DW      8000H
WORD2   DW      4000H
; -------------------------------------------
MAIN    PROC    NEAR              ;Основная процедура:
      CALL    C10MUL            ;Вызвать умнож. MUL
      CALL    D10IMUL           ;Вызвать умнож. IMUL
      RET
MAIN    ENDP
;             Пример умножения MUL:
;             --------------------
C10MUL  PROC
      MOV     AL,BYTE1          ;Байт * байт
      MUL     BYTE2            ; произведение в AX
      MOV     AX,WORD1          ;Слово * слово
      MUL     WORD2             ; произведение в DX:AX
      MOV     AL,BYTE1          ;Байт * слово
      SUB     AH,AH            ; расшир. множ. в AH
      MUL     WORD1             ; произведение в DX:AX
```

```
     RET
C10MUL  ENDP
;            Пример умножения IMUL:
;            ---------------------
D10IMUL PROC
     MOV     AL,BYTE1          ;Байт * байт
     IMUL    BYTE2            ; произведение в AX
     MOV     AX,WORD1           ;Слово * слово
     IMUL    WORD2             ; произвед. в DX:AX
     MOV     AL,BYTE1          ;Байт * слово
     CBW                     ; расшир. множ. в AH
     IMUL    WORD1             ; произвед. в DX:AX
     RET
D10IMUL ENDP

CODESG  ENDS
     END     BEGIN
```

Рис.12.3. Беззнаковое и знаковое умножение.

Второй пример команды MUL генерирует шест.10000000 в регистрах DX:AX.
Третий пример команды MUL выполняет умножение слова на байт и требует расширение байта BYTE1 до размеров слова. Так как предполагаются беззнаковые величины, то в примере левый бит регистра AH равен нулю. (При использовании команды CBW значение левого бита регистра AL может быть 0 или 1). Произведение - шест.00400000 получается в регистрах DX:AX.

Знаковое умножение: Команда IMUL

Команда IMUL (Integer MULtiplication - умножение целых чисел) умножает знаковые числа. На рис.12.3 в процедуре D10IMUL используются те же три примера умножения, что и в процедуре C10MUL, но вместо команд MUL

записаны команды IMUL.
Первый пример команды IMUL умножает шест.80 (отрицательное число) на шест.40 (положительное число). Произведение - шест.E000 получается в регистре AX. Используя те же данные, команда MUL дает в результате шест.2000, так что можно видеть разницу в использовании команд MUL и IMUL. Команда MUL рассматривает шест.80 как +128, а команда IMUL - как -128. В результате умножения -128 на +64 получается -8192 или шест.E000. (Попробуйте преобразовать шест.E000 в десятичный формат).
Второй пример команды IMUL умножает шест.8000 (отрицательное значение) на шест.2000 (положительное значение). Произведение - шест.F0000000 получается в регистрах DX:AX и представляет собой отрицательное значение.
Третий пример команды IMUL перед умножением выполняет расширение байта BYTE1 до размеров слова в регистре AX. Так как значения предполагаются знаковые, то в примере используется команда CBW для перевода левого знакового бита в регистр AH: шест.80 в регистре AL превращается в шест.FF80 в регистре AX. Поскольку множитель в слове WORD1 имеет также отрицательное значение, то произведение должно получится положительное. В самом деле: шест.00400000 в регистрах DX:AX - такой же результат, как и в случае умножения командой MUL, которая предполагала положительные сомножители.
Таким образом, если множимое и множитель имеет одинаковый знаковый бит, то команды MUL и IMUL генерируют одинаковый результат. Но, если сомножители имеют разные знаковые биты, то команда MUL вырабатывает положительный результат умножения, а команда IMUL - отрицательный.

Можно обнаружить это, используя отладчик DEBUG для трассировки примеров.

П о в ы ш е н и е э ф ф е к т и в н о с т и у м н о ж е н и я: При умножении на степень числа 2 (2,4,8 и т.д.) более эффективным является сдвиг влево на требуемое число битов. Сдвиг более чем на 1 требует загрузки величины сдвига в регистр CL. В следующих примерах предположим, что множимое находится в регистре AL или AX:

```
Умножение на 2:     SHL  AL,1
Умножение на 8:     MOV  CL,3
                    SHL  AX,CL
```

Многословное умножение

Обычно умножение имеет два типа: "байт на байт" и "слово на слово". Как уже было показано, максимальное знаковое значение в слове ограничено величиной +32767. Умножение больших чисел требует выполнения некоторых дополнительных действий. Рассматриваемый подход предполагает умножение каждого слова отдельно и сложение полученных результатов. Рассмотрим следующее умножение в десятичном формате:

```
 1365
  x12
-----
 2730
1365
-----
16380
```

Представим, что десятичная арифметика может умножать только двузначные числа. Тогда можно умножить 13 и 65 на 12 раздельно, следующим образом:

```
 13      65
x12     x12
---     ---
 26     130
13      65
---     ---
156     780
```

Следующим шагом сложим полученные произведения, но поскольку число 13 представляло сотни, то первое произведение в действительности будет 15600:

```
15600
 +780
-----
16380
```

Ассемблерная программа использует аналогичную технику за исключением того, что данные имеют размерность слов (четыре цифры) в шестнадцатеричном формате.

У м н о ж е н и е д в о й н о г о с л о в а н а с л о в о. Процедура E10XMUL на рис.12.4 умножает двойное слово на слово. Множимое, MULTCND, состоит из двух слов, содержащих соответственно шест.3206 и шест.2521. Определение данных в виде двух слов (DW) вместо двойного слова (DD) обусловлено необходимостью правильной адресации для команд MOV, пересылающих слова в регистр AX. Множитель MULTPLR содержит шест.6400. Область для записи произведения, PRODUCT, состоит из трех слов. Первая

команда MUL перемножает MULTPLR и правое слово поля MULTCND; произведение - шест.0E80 E400 записывается в PRODUCT+2 и PRODUCT+4. Вторая команда MUL перемножает MULTPLR и левое слово поля MULTCND, получая в результате шест. 138A 5800. Далее выполняется сложение двух произведений следующим образом:

```
          Произведение 1:    0000 0E80 E400
          Произведение 2:    138A 5800
                                  -------------
          Результат:         138A 6680 E400
```

Так как первая команда ADD может выработать перенос, то второе сложение выполняется командой сложения с переносом ADC (ADd with Carry). В силу обратного представления байтов в словах в процессорах 8086/8088, область PRODUCT в действительности будет содержать значение 8A13 8066 00E4. Программа предполагает, что первое слово в области PRODUCT имеет начальное значение 0000.

```
TITLE   EXDWMUL - Умножение двойных слов
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
        ORG     100H
BEGIN:  JMP     SHORT MAIN
; --------------------------------------------
MULTCND DW      3206H           ;Элементы данных
        DW      2521H
MULTPLR DW      6400H
        DW      0A26H
PRODUCT DW      0
        DW      0
        DW      0
        DW      0
; --------------------------------------------
MAIN    PROC    NEAR            ;Основная процедура
        CALL    E10XMUL         ;Вызвать 1-е умножение
        CALL    Z10ZERO         ;Очистить произведение
        CALL    F10XMUL         ;Вызвать 2-е умножение
        RET
MAIN    ENDP
;               Умножение двойного слова на слово:
; ---------------------------------------------
E10XMUL PROC
        MOV     AX,MULTCND+2    ;Умножить правое слова
        MUL     MULTPLR         ; множимого
        MOV     PRODUCT+4,AX    ;Записать произведение
        MOV     PRODUCT+2,DX

        MOV     AX,MULTCND      ;Умножить левое слово
        MUL     MULTPLR         ; множимого
        ADD     PRODUCT+2,AX    ;Сложить с полученным ранее
        ADC     PRODUCT,DX
        RET
E10XMUL ENDP
;               Перемножение двух двойных слов:
; -------------------------------------------
F10XMUL PROC
        MOV     AX,MULTCND+2    ;Слово-2 множимого
        MUL     MULTPLR+2       ; * слово-2 множителя
        MOV     PRODUCT+6,AX    ;Сохранить результат
        MOV     PRODUCT+4,DX
```

```
       MOV     AX,MULTCND+2    ;Слово-2 множимого
       MUL     MULTPLR         ; * слово-1 множителя
       ADD     PRODUCT+4,AX    ;Сложить с предыдущим
       ADC     PRODUCT+6,DX
       ADC     PRODUCT,00      ;Прибавить перенос

       MOV     AX,MULTCND      ;Слово-1 множимого
       MUL     MULTPLR+2       ; * слово-2 множителя
       ADD     PRODUCT+4,AX    ;Сложить с предыдущим
      ADC     PRODUCT+6,DX
       ADC     PRODUCT,00      ;Прибавить перенос
       MOV     AX,MULTCND      ;Слово-1 множимого
       MUL     MULTPLR         ; * слово-1 множителя
       ADD     PRODUCT+2,AX    ;Сложить с предыдущим
       ADC     PRODUCT,DX
       RET
F10XMUL ENDP
;              Очистка области результата:
; --------------------------------------
Z10XMUL PROC
       MOV     PRODUCT,0000
       MOV     PRODUCT+2,0000
       MOV     PRODUCT+4,0000
       MOV     PRODUCT+6,0000
       RET
Z10XMUL ENDP

CODESG  ENDS
       END     BEGIN
```

Рис.12.4. Многословное умножение.

У м н о ж е н и е "д в о й н о г о с л о в а н а д в о й н о е с л о в о". Умножение двух двойных слов включает следующие четыре операции умножения:

```
          Множимое      Множитель

          слово 2   x    слово 2
          слово 2   x    слово 1
          слово 1   x    слово 2
          слово 1   x    слово 1
```

Каждое произведение в регистрах DX и AX складывается с соответствующим словом в окончательном результате. Пример такого умножения приведен в процедуре F10XMUL на рис.12.4. Множимое MULTCND содержит шест.3206 2521, множитель MULTPLR - шест.6400 0A26. Результат заносится в область PRODUCT, состоящую из четырех слов.

Хотя логика умножения двойных слов аналогична умножению двойного слова на слово, имеется одна особенность, после пары команд сложения ADD/ADC используется еще одна команда ADC, которая прибавляет 0 к значению в поле PRODUCT. Это необходимо потому, что первая команда ADC сама может вызвать перенос, который последующие команды могут стереть. Поэтому вторая команда ADC прибавит 0, если переноса нет, и прибавит 1, если перенос есть. Финальная пара команд ADD/ADC не требует дополнительной команды ADC, так как область PRODUCT достаточно велика для генерации окончательного результата и переноса на последнем этапе не будет.

Окончательный результат 138A 687C 8E5C CCE6 получится в поле PRODUCT

в обратной записи байт в словах. Выполните трассировку этого примера с помощью отладчика DEBUG.

СДВИГ РЕГИСТРОВОЙ ПАРЫ DX:AX

Следующая подпрограмма может быть полезна для сдвига содержимого регистровой пары DX:AX вправо или влево. Можно придумать более эффективный метод, но данный пример представляет общий подход для любого числа циклов (и, соответственно, сдвигов) в регистре CX. Заметьте, что сдвиг единичного бита за разрядную сетку устанавливает флаг переноса.

```
          Сдвиг влево на 4 бита
      MOV  CX,04     ;Инициализация на 4 цикла
  C20: SHL  DX,1      ;Сдвинуть DX на 1 бит влево
      SHL  AX,1      ;Сдвинуть AX на 1 бит влево
      ADC  DX,00     ;Прибавить значение переноса
      LOOP C20       ;Повторить
          Сдвиг вправо на 4 бита
      MOV  CX,04     ;Инициализация на 4 цикла
  D20: SHR  AX,1      ;Сдвинуть AX на 1 бит вправо
      SHR  DX,1      ;Сдвинуть DX на 1 бит вправо
      JNC  D30       ;Если есть перенос,
      OR   AH,10000000B  ; то вставить 1 в AH
  D30: LOOP D20       ;Повторить
```

Ниже приведен более эффективный способ для сдвига влево, не требующий организации цикла. В этом примере фактор сдвига записывается в регистр CL. Пример написан для сдвига на 4 бита, но может быть адаптирован для других величин сдвигов:

```
      MOV  CL,04     ;Установить фактор сдвига
      SHL  DX,CL     ;Сдвинуть DX влево на 4 бита
      MOV  BL,AH     ;Сохранить AH в BL
      SHL  AX,CL     ;Сдвинуть AX влево на 4 бита
      SHL  BL,CL     ;Сдвинуть BL вправо на 4 бита
      OR   DL,BL     ;Записать 4 бита из BL в DL
```

ДЕЛЕНИЕ

Операция деления для беззнаковых данных выполняется командой DIV, а для знаковых - IDIV. Ответственность за подбор подходящей команды лежит на программисте. Существуют две основные операции деления:

Д е л е н и е "с л о в а н а б а й т". Делимое находится в регистре AX, а делитель - в байте памяти или а однобайтовом регистре. После деления остаток получается в регистре AH, а частное - в AL. Так как однобайтовое частное очень мало (максимально +255 (шест.FF) для беззнакового деления и +127 (шест.7F) для знакового), то данная операция имеет ограниченное использование.

```
               |  AX   |          |  AH   |  AL   |
  До деления: |Делимое|    После: |Остаток|Частное|
```

Д е л е н и е "д в о й н о г о с л о в а н а с л о в о". Делимое находится в регистровой паре DX:AX, а делитель - в слове памяти или а регистре. После деления остаток получается в регистре DX, а частное в регистре AX. Частное в одном слове допускает максимальное значение +32767

(шест.FFFF) для беззнакового деления и +16383 (шест.7FFF) для знакового.

```
                |   DX   ||   AX   |          |  AH   ||  AL   |
      До деления: |Ст.часть||Мл.часть|  После: |Остаток||Частное|
                |     Делимое      |
```

В единственном операнде команд DIV и IDIV указывается делитель. Рассмотрим следующую команду:

```
            DIV  DIVISOR
```

Если поле DIVISOR определено как байт (DB), то операция предполагает деление слова на байт. Если поле DIVISOR определено как слово (DW), то операция предполагает деление двойного слова на слово.

При делении, например, 13 на 3, получается результат 4 1/3. Частное есть 4, а остаток - 1. Заметим, что ручной калькулятор (или программа на языке BASIC) выдает в этом случае результат 4,333.... Значение содержит целую часть (4) и дробную часть (,333). Значение 1/3 и 333... есть дробные части, в то время как 1 есть остаток от деления.

Беззнаковое деление: Команда DIV

Команда DIV делит беззнаковые числа. На рис.12.5 в процедуре D10DIV дано четыре примера деления: слово на байт, байт на байт, двойное слово на слово и слово на слово. Первый пример команды DIV делит шест.2000 (8092) на шест.80 (128). В результате остаток 00 получается в регистре AH, а частное шест.40 (64) - в регистре AL.

Второй пример команды DIV выполняет прежде расширение байта BYTE1 до размеров слова. Так как здесь предполагается беззнаковая величина, то в примере левый бит регистра AH равен нулю. В результате деления остаток - шест.12 получается в регистре AH, а частное шест.05 - в регистре AL.

Третий пример команды DIV генерирует остаток шест.1000 в регистре DX и частное шест.0080 в регистре AX.

В четвертом примере команды DIV сначала выполняется расширение слова WORD1 до двойного слова в регистре DX. После деления остаток шест.0000 получится в регистре DX, а частное шест.0002 - в регистре AX.

```
       page    60,132
TITLE   EXDIV   (COM) Пример операций DIV и IDIV
CODESG  SEGMENT PARA 'Code'
       ORG     100H
BEGIN:  JMP     SHORT MAIN
; ---------------------------------------------
BYTE1   DB      80H           ;Data items
BYTE2   DB      16H
WORD1   DW      2000H
WORD2   DW      0010H
WORD3   DW      1000H
; ---------------------------------------------
MAIN    PROC    NEAR          ;Основная процедура
       CALL    D10DIV        ;Вызов подпрограммы DIV
       CALL    E10IDIV       ;Вызов подпрограммы IDIV
MAIN    ENDP
;             Примеры с командой DIV:
; ---------------------------------------------
D10DIV  PROC
       MOV     AX,WORD1      ;Слово / байт
       DIV     BYTE1         ; остаток:частное в AH:AL
```

```
       MOV     AL,BYTE1      ;Байт / байт
       SUB     AH,AH         ; расширить делимое в AH
       DIV     BYTE3         ; остаток:частное в AH:AL

       MOV     DX,WORD2      ;Двойное слово / слово
       MOV     AX,WORD3      ; делимое в DX:AX
       DIV     WORD1         ; остаток:частное в DX:AX
       MOV     AX,WORD1      ;Слово / слово
       SUB     DX,DX         ; расширить делимое в DX
       DIV     WORD3         ; остаток:частное в DX:AX
       RET
D10DIV  ENDP
;              Примеры с командой IDIV:
; -------------------------------------------
E10IDIV PROC
       MOV     AX,WORD1      ;Слово / байт
       IDIV    BYTE1         ; остаток:частное в AH:AL
       MOV     AL,BYTE1      ;Байт / байт
       CBW                   ; расширить делимое в AH
       IDIV    BYTE3         ; остаток:частное в AH:AL

       MOV     DX,WORD2      ;Двойное слово / слово
       MOV     AX,WORD3      ; делимое в DX:AX
       IDIV    WORD1         ; остаток:частное в DX:AX
       MOV     AX,WORD1      ;Слово / слово
       CWD                   ; расширить делимое в DX
       IDIV    WORD3         ; остаток:частное в DX:AX
       RET
E10DIV  ENDP
CODESG  ENDS
       END     BEGIN
```

Рис.12.5. Беззнаковое и знаковое деление.

Знаковое деление: Команда IDIV

Команда IDIV (Integer DIVide) выполняет деление знаковых чисел. На рис.12.5 в процедуре E10IDIV используются те же четыре примера деления, что и в процедуре D10DIV, но вместо команд DIV записаны команды IDIV. Первый пример команды IDIV делит шест.2000 (положительное число) на шест.80 (отрицательное число). Остаток от деления - шест. 00 получается в регистре AH , а частное - шест. C0 (-64) - в регистре AL. Команда DIV, используя те же числа, генерирует частное +64.

Шестнадцатиричные результаты трех остальных примеров деления приведены ниже:

Пример команды IDIV	Остаток	Частное
2	EE (-18)	FB (-5)
3	1000 (4096)	0080 (128)
4	0000	0002

Только в примере 4 вырабатывается такой же результат, что и для команды DIV. Таким образом, если делимое и делитель имеют одинаковый знаковый бит, то команды DIV и IDIV генерируют одинаковый результат. Но, если делимое и делитель имеют разные знаковые биты, то команда DIV генерирует положительное частное, а команда IDIV - отрицательное частное. Можно обнаружить это, используя отладчик DEBUG для трассировки этих примеров.

Повышение производительности. При делении на степень числа 2 (2, 4, и

т.д.) более эффективным является сдвиг вправо на требуемое число битов. В следующих примерах предположим, что делимое находится в регистре AX:

```
Деление на 2:      SHR  AX,1

Деление на 8:      MOV  CL,3
              SHR  AX,CL
```

Переполнения и прерывания

Используя команды DIV и особенно IDIV, очень просто вызвать переполнение. Прерывания приводят (по крайней мара в системе, используемой при тестировании этих программ) к непредсказуемым результатам. В операциях деления предполагается, что частное значительно меньше, чем делимое. Деление на ноль всегда вызывает прерывание. Но деление на 1 генерирует частное, которое равно делимому, что может также легко вызвать прерывание.

Рекомендуется использовать следующее правило: если делитель - байт, то его значение должно быть меньше, чем левый байт (AH) делителя: если делитель - слово, то его значение должно быть меньше, чем левое слово (DX) делителя. Проиллюстрируем данное правило для делителя, равного 1:

Операция деления:	Делимое	Делитель	Частное
Слово на байт:	0123	01	(1)23
Двойное слово на слово:	0001 4026	0001	(1)4026

В обоих случаях частное превышает возможный размер. Для того чтобы избежать подобных ситуаций, полезно вставлять перед командами DIV и IDIV соответствующую проверку. В первом из следующих примеров предположим, что DIVBYTE - однобайтовый делитель, а делимое находится уже в регистре AX. Во втором примере предположим, что DIVWORD - двухбайтовый делитель, а делимое находится в регистровой паре DX:AX.

```
Слово на байт          Двойное слово на байт

CMP  AH,DIVBYTE          CMP  DX,DIVWORD
JNB  переполнение        JNB  переполнение
DIV  DIVBYTE             DIV  DIVWORD
```

Для команды IDIV данная логика должна учитывать тот факт, что либо делимое, либо делитель могут быть отрицательными, а так как сравниваются абсолютные значения, то необходимо использовать команду NEG для временного перевода отрицательного значения в положительное.

Деление вычитанием

Если частное слишком велико, то деление можно выполнить с помощью циклического вычитания. Метод заключается в том, что делитель вычитается из делимого и в этом же цикле частное увеличивается на 1. Вычитание продолжается, пока делимое остается больше делителя. В следующем примере, делитель находится в регистре AX, а делимое - в BX, частное вырабатывается в CX:

```
    SUB  CX,CX     ;Очистка частного
C20: CMP  AX,BX     ;Если делимое < делителя,
    JB   C30       ; то выйти
    SUB  AX,BX     ;Вычитание делителя из делимого
    INC  CX        ;Инкремент частного
    JMP  C20       ;Повторить цикл
```

```
        C30: RET          ;Частное в CX, остаток в AX
```

В конце подпрограммы регистр CX будет содержать частное, а AX - остаток. Пример умышленно примитивен для демонстрации данной техники деления. Если частное получается в регистровой паре DX:AX, то необходимо сделать два дополнения:

1. В метке C20 сравнивать AX и BX только при нулевом DX.
2. После команды SUB вставить команду SBB DX,00.

П р и м е ч а н и е: очень большое частное и малый делитель могут вызвать тысячи циклов.

ПРЕОБРАЗОВАНИЕ ЗНАКА

Команда NEG обеспечивает преобразование знака двоичных чисел из положительного в отрицательное и наоборот. Практически команда NEG устанавливает противоположные значения битов и прибавляет 1. Примеры:

```
        NEG  AX
        NEG  BL
        NEG  BINAMT    ;(байт или слово в памяти)
```

Преобразование знака для 35-битового (или большего) числа включает больше шагов. Предположим, что регистровая пара DX:AX содержит 32-битовое двоичное число. Так как команда NEG не может обрабатывать два регистра одновременно, то ее использование приведет к неправильному результату. В следующем примере показано использование команды NOT:

```
        NOT  DX   ;Инвертирование битов
        NOT  AX   ;Инвертирование битов
        ADD  AX,1 ;Прибавление 1 к AX
        ADC  DX,0 ;Прибавление переноса к DX
```

Остается одна незначительная проблема: над числами, представленными в двоичном формате, удобно выполнять арифметические операции, если сами числа определены в программе. Данные, вводимые в программу с дискового файла, могут также иметь двоичный формат. Но данные, вводимые с клавиатуры, представлены в ASCII-формате. Хотя ASCII-коды удобны для отображения и печати, они требуют специальных преобразований в двоичный формат для арифметических вычислений. Но это уже тема следующей главы.

ПРОЦЕССОРЫ INTEL 8087 И 80287 ДЛЯ ОБРАБОТКИ ЧИСЛОВЫХ ДАННЫХ

Системная плата компьютера содержит пустое гнездо, зарезервированное для числового процессора Intel 8087 (или 80287). Сопроцессор 8087 действует совместно с 8088, а сопроцессор 80287 действует совместно с 80286. Каждый сопроцессор имеет собственный набор команд и средства для операций с плавающей запятой для выполнения экспоненциальных, логарифмических и тригонометрических функций. Сопроцессор содержит восемь 80-битовых регистров с плавающей запятой, которые могут представить числовые значения до 10 в 400 степени. Математические вычисления в сопроцессоре выполняются примерно в 100 раз быстрее, чем в основном процессоре.

Основной процессор выполняет специальные операции и передает числовые данные в сопроцессор, который выполняет необходимые вычисления и возвращает результат. Для ассемблирования с помощью транслятора MASM,

необходимо добавлять параметр /E или /R, например, MASM /R.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Будьте особенно внимательны при использовании однобайтовых регистров. Знаковые значения здесь могут быть от -128 до +127.

- Для многословного сложения используйте команду ADC для учета переносов от предыдущих сложений. Если операция выполняется в цикле, то используя команду CLC, установите флаг переноса в 0.

- Используйте команды MUL или DIV для беззнаковых данных и команды IMUL или IDIV для знаковых.

- При делении будьте осторожны с переполнениями. Если нулевой делитель возможен, то обеспечьте проверку этой операции. Кроме того, делитель должен быть больше содержимого регистра AH (для байта) или DX (для слова).

- Для умножения или деления на степень двойки используйте сдвиг. Сдвиг вправо выполняется командой SHR для беззнаковых полей и командой SAR для знаковых полей. Для сдвига влево используются идентичные команды SHL и SAL.

- Будьте внимательны при ассемблировании по умолчанию. Например, если поле FACTOR определено как байт (DB), то команда MUL FACTOR полагает множимое в регистре AL, а команда DIV FACTOR полагает делимое в регистре AX. Если FACTOR определен как слово (DW), то команда MUL FACTOR полагает множимое в регистре AX, а команда DIV FACTOR полагает делимое в регистровой паре DX:AX.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

Все вопросы имеют отношение к следующим данным:

```
DATAX     DW   0148H
          DW   2316H
DATAY     DW   0237H
          DW   4052H
```

12.1. Закодируйте команды для сложения а) слова DATAX со словом DATAY; б) двойного слова, начинающегося по адресу DATAX, с двойным словом в DATAY.

12.2. Объясните действие следующих команд:

```
STC
MOV BX,DATAX
ADC BX,DATAY
```

12.3. Закодируйте команды для умножения (MUL): а) слова DATAX на слово DATAY; б) двойного слова , начинающегося по адресу DATAX, на слово DATAY.

12.4. Какой делитель, кроме нуля, вызывает ошибку переполнения?

12.5. Закодируйте команды для деления (DIV): а) слова DATAX на 23; б)

двойного слова, начинающегося по адресу DATAX, на слово DATAY.

12.6. Последний пример в разделе "Сдвиг регистровой пары DX:AX" является более эффективным по сравнению с предыдущими примерами для сдвига влево на четыре бита. Измените пример для сдвига вправо на четыре бита.

ГЛАВА 13 Арифметические операции II: Обработка данных в форматах ASCII и BCD

Ц е л ь: Рассмотреть ASCII и BCD форматы данных и дать сведения о преобразованиях между этими форматами и двоичным форматом.

ВВЕДЕНИЕ

Для получения высокой производительности компьютер выполняет арифметические операции над числами в двоичном формате. Как показано в гл.12, этот формат не вызывает особых трудностей, если данные определены в самой программе. Во многих случаях новые данные вводятся программой с клавиатуры в виде ASCII символов в десятичном формате. Аналогично вывод информации на экран осуществляется в кодах ASCII. Например, число 23 в двоичном представлении выглядит как 00010111 или шест.17; в коде ASCII на каждый символ требуется один байт и число 25 в ASCII-коде имеет внутреннее представление шест.3235.

Назначение данной главы - показать технику преобразования данных из ASCII-формата в двоичный формат для выполнения арифметических операций и обратного преобразования двоичных результатов в ASCII-формат для вывода на экран или принтер. Программа, приведенная в конце главы , демонстрирует большую часть материала гл.1 - 12.

При программировании на языках высокого уровня, таких как BASIC или Pascal, для обозначения порядка числа или положения десятичной запятой (точки) можно положиться на компилятор. Однако, компьютер не распознает десятичную запятую (точку) в арифметических полях. Так как двоичные числа не имеют возможности установки десятичной (или двоичной) запятой (точки), то именно программист должен подразумевать и определить порядок обрабатываемых чисел.

ASCII-ФОРМАТ

Данные, вводимые с клавиатуры, имеют ASCII-формат, например, буквы SAM имеют в памяти шестнадцатиричное представление 53414D, цифры 1234 - шест.31323334. Во многих случаях формат алфавитных данных, например, имя человека или описание статьи, не меняется в программе. Но для выполнения арифметических операций над числовыми значениями, такими как шест.31323334, требуется специальная обработка.

С помощью следующих ассемблерных команд можно выполнять арифметические операции непосредственно над числами в ASCII-формате:

AAA (ASCII Adjust for Addition - коррекция для сложения ASCII-кода)
AAD (ASCII Adjust for Division - коррекция для деления ASCII-кода)
AAM (ASCII Adjust for Multiplication - коррекция для умножения ASCII-кода)
AAS (ASCII Adjust for Subtraction - коррекция для вычитания ASCII-кода)

Эти команды кодируются без операндов и выполняют автоматическую коррекцию в регистре AX. Коррекция необходима, так как ASCII-код представляет так

называемый распакованный десятичный формат, в то время, как компьютер выполняет арифметические операции в двоичном формате.

Сложение в ASCII-формате

Рассмотрим процесс сложения чисел 8 и 4 в ASCII-формате:

```
Шест.   38
        34
        --
Шест.   6C
```

Полученная сумма неправильна ни для ASCII-формата, ни для двоичного формата. Однако, игнорируя левую 6 и прибавив 6 к правой шест.C: шест.C + 6 = шест.12 - получим правильный результат в десятичном формате. Правильный пример слегка упрощен, но он хорошо демонстрирует процесс, который выполняет команда AAA при коррекции.

В качестве примера, предположим, что регистр AX содержит шест.0038, а регистр BX - шест.0034. Числа 38 и 34 представляют два байта в ASCII-формате, которые необходимо сложить. Сложение и коррекция кодируется следующими командами:

```
ADD  AL,BL     ;Сложить 34 и 38
AAA            ;Коррекция для сложения ASCII-кодов
```

Команда AAA проверяет правую шест. цифру (4 бита) в регистре AL. Если эта цифра находится между A и F или флаг AF равен 1, то к регистру AL прибавляется 6, а к регистру AH прибавляется 1, флаги AF и CF устанавливаются в 1. Во всех случаях команда AAA устанавливает в 0 левую шест. цифру в регистре AL. Результат - в регистре AX:

```
После команды ADD:  006C
После команды AAA:  0102
```

Для того, чтобы выработать окончательное ASCII-представление, достаточно просто поставить тройки на место левых шест. цифр:

```
OR   AX,3030H  ;Результат 3132
```

Все показанное выше представляет сложение однобайтовых чисел. Сложение многобайтовых ASCII-чисел требует организации цикла, который выполняет обработку справа налево с учетом переноса. Пример , показанный на рис.13.1 складывает два трехбайтовых ASCII-числа в четырехбайтовую сумму. Обратите внимание на следующее:

```
TITLE   ASCADD  (COM) Сложение чисел в ASCII-формате
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
        ORG     100H
BEGIN:  JMP     SHORT MAIN
; ---------------------------------------------
ASC1    DB      '578'       ;Элементы данных
ASC2    DB      '694'
ASC3    DB      '0000'
; ---------------------------------------------
MAIN    PROC    NEAR
        CLC
        LEA     SI,AASC1+2   ;Адреса ASCII-чисел
```

```
       LEA     DI,AASC2+2
       LEA     BX,AASC1+3
       MOV     CX,03        ;Выполнить 3 цикла
A20:
       MOV     AH,00        ;Очистить регистр AH
       MOV     AL,[SI]      ;Загрузить ASCII-байт
       ADC     AL,[DI]      ;Сложение (с переносом)
       AAA                  ;Коррекция для ASCII
       MOV     [BX],AL      ;Сохранение суммы
       DEC     SI
       DEC     DI
       DEC     BX
       LOOP    A20          ;Циклиться 3 раза
       MOV     [BX],AH      ;Сохранить перенос
       RET
MAIN    ENDP
CODESG  ENDS
       END     BEGIN
```
____

Рис.13.1. Сложение в ASCII-формате.

- В программе используется команда ADC, так как любое сложение может вызвать перенос, который должен быть прибавлен к следующему (слева) байту. Команда CLC устанавливает флаг CF в нулевое состояние.
- Команда MOV очищает регистр AH в каждом цикле, так как команда AAA может прибавить к нему единицу. Команда ADC учитывает переносы. Заметьте, что использование команд XOR или SUB для очистки регистра AH изменяет флаг CF.
- Когда завершается каждый цикл, происходит пересылка содержимого регистра AH (00 или 01) в левый байт суммы.
- В результате получается сумма в виде 01020702. Программа не использует команду OR после команды AAA для занесения левой тройки, так как при этом устанавливается флаг CF, что изменит результат команды ADC. Одним из решений в данном случае является сохранение флагового регистра с помощью команды PUSHF, выполнение команды OR, и, затем, восстановление флагового регистра командой POPF:

```
ADC  AL,[DI]   ;Сложение с переносом
AAA            ;Коррекция для ASCII
PUSHF          ;Сохранение флагов
OR   AL,30H    ;Запись левой тройки
POPF           ;Восстановление флагов
MOV  [BX],AL   ;Сохранение суммы
```

Вместо команд PUSHF и POPF можно использовать команды LAHF (Load AH with Flags - загрузка флагов в регистр AH) и SAHF (Store AH in Flag register - запись флагов из регистра AH во флаговый регистр). Команда LAHF загружает в регистр AH флаги SF, ZF, AF, PF и CF; а команда SAHF записывает содержимое регистра AH в указанные флаги. В приведенном примере, однако, регистр AH уже используется для арифметических переполнений. Другой способ вставки троек для получения ASCII-кодов цифр - организовать обработку суммы командой OR в цикле.

Вычитание в ASCII-формате

Команда AAS (ASCII Adjust for Subtraction - коррекция для вычитания ASCII-кодов) выполняется аналогично команде AAA. Команда AAS проверяет правую шест. цифру (четыре бита) в регистре AL. Если эта цифра лежит между

A и F или флаг AF равен 1, то из регистра AL вычитается 6, а из регистра AH вычитается 1, флаги AF и CF устанавливаются в 1. Во всех случаях команда AAS устанавливает в 0 левую шест.цифру в регистре AL.

В следующих двух примерах предполагается, что поле ASC1 содержит шест.38, а поле ASC2 - шест.34:

```
          Пример 1:            AX     AF

             MOV  AL,ASC1       ;0038
             SUB  AL,ASC2       ;0034    0
             AAS             ;0004    0


          Пример 2:            AX     AF

             MOV  AL,ASC2       ;0034
             SUB  AL,ASC1       ;00FC    1
             AAS             ;FF06    1
```

В примере 1 команде AAS не требуется выполнять коррекцию. В примере 2, так как правая цифра в регистре AL равна шест.C, команда AAS вычитает 6 из регистра AL и 1 из регистра AH и устанавливает в 1 флаги AF и CF. Результат (который должен быть равен -4) имеет шест. представление FF06, т.е. десятичное дополнение числа -4.

Умножение в ASCII-формате

Команда AAM (ASCII Adjust for Multiplication - коррекция для умножения ASCII-кодов) выполняет корректировку результата умножения ASCII-кодов в регистре AX. Однако, шест. цифры должны быть очищены от троек и полученные данные уже не будут являться действительными ASCII-кодами. (В руководствах фирмы IBM для таких данных используется термин распакованный десятичный формат). Например, число в ASCII-формате 31323334 имеет распакованное десятичное представление 01020304. Кроме этого, надо помнить, что коррекция осуществляется только для одного байта за одно выполнение, поэтому можно умножать только одно-байтовые поля; для более длинных полей необходима организация цикла.

Команда AAM делит содержимое регистра AL на 10 (шест.0A) и записывает частное в регистр AH, а остаток в AL. Предположим, что в регистре AL содержится шест.35, а в регистре CL - шест.39. Следующие команды умножают содержимое регистра AL на содержимое CL и преобразуют результат в ASCII-формат:

```
                                              AX:
          AND  CL,0FH    ;Преобразовать CL в 09
          AND  AL,0FH    ;Преобразовать AL в 05        0005
          MUL  CL        ;Умножить AL на CL            002D
          AAM            ;Преобразовать в распак.дес.  0405
          OR   AX,3030H  ;Преобразовать в ASCII-ф-т    3435
```

Команда MUL генерирует 45 (шест.002D) в регистре AX, после чего команда AAM делит это значение на 10, записывая частное 04 в регистр AH и остаток 05 в регистр AL. Команда OR преобразует затем распакованное десятичное число в ASCII-формат.

Пример на рис.13.2 демонстрирует умножение четырехбайтового множимого на однобайтовый множитель. Так как команда AAM может иметь дело только с однобайтовыми числами, то в программе организован цикл, который обрабатывает байты справа налево. Окончательный результат умножения в данном примере - 0108090105.

Если множитель больше одного байта, то необходимо обеспечить еще один

цикл, который обрабатывает множитель. В этом случае проще будет преобразовать число из ASCII-формата в двоичный формат (см. следующий раздел "Преобразование ASCII-формата в двоичный формат").

```
TITLE   ASCMUL  (COM) Умножение ASCII-чисел
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
        ORG     100H
BEGIN:  JMP     MAIN
; ------------------------------------------
MULTCND DB      '3783'          ;Элементы данных
MULTPLR DB      '5'
PRODUCT DB      5 DUP(0)
; ------------------------------------------
MAIN    PROC    NEAR
        MOV     CX,04           ;4 цикла
        LEA     SI,MULTCND+3
        LEA     DI,PRODUCT+4
        AND     MULTPLR,0FH     ;Удалить ASCII-тройку
A20:
        MOV     AL,[SI]         ;Загрузить ASCII-символ
                                ; (можно LODSB)
        AND     AL,OFH          ;Удалить ASCII-тройку
        MUL     MULTPLR         ;Умножить
        AAM                     ;Коррекция для ASCII
        ADD     AL,[DI]         ;Сложить с
        AAA                     ; записанным
        MOV     [DI],AL         ; произведением
        DEC     DI
        MOV     [DI],AH         ;Записать перенос
        DEC     SI
        LOOP    A20             ;Циклиться 4 раза
        RET
MAIN    ENDP
CODESG  ENDS
        END     BEGIN
```

Рис.13.2. Умножение в ASCII-формате.

Деление в ASCII-формате

Команда AAD (ASCII Adjust for Division - коррекция для деления ASCII-кодов) выполняет корректировку ASCII-кода делимого до непосредственного деления. Однако, прежде необходимо очистить левые тройки ASCII-кодов для получения распакованного десятичного формата. Команда AAD может оперировать с двухбайтовыми делимыми в регистре AX. Предположим, что регистр AX содержит делимое 3238 в ASCII-формате и регистр CL содержит делитель 37 также в ASCII-формате. Следующие команды выполняют коррекцию для последующего деления:

```
                                                AX:
    AND  CL,0FH    ;Преобразовать CL в распак.дес.
    AND  AX,0F0FH  ;Преобразовать AX в распак.дес.    0208
    AAD            ;Преобразовать в двоичный          001C
    DIV  CL        ;Разделить на 7                    0004
```

Команда AAD умножает содержимое AH на 10 (шест.0A), прибавляет результат 20 (шест.14) к регистру AL и очищает регистр AH. Значение 001C есть шест.

представление десятичного числа 28. Делитель может быть только однобайтовый от 01 до 09.

Пример на рис.13.3 выполняет деление четырехбайтового делимого на однобайтовый делитель. В программе организован цикл обработки делимого справа налево. Остатки от деления находятся в регистре AH и команда AAD корректирует их в регистре AL. Окончательный результат: частное 00090204 и в регистре AH остаток 02.

Если делитель больше одного байта, то необходимо построить другой цикл для обработки делителя, но лучше воспользоваться следующим разделом "Преобразование ASCII-формата в двоичный формат."

```
TITLE   ASCDIV  (COM) Деление ASCII-чисел
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
        ORG     100H

BEGIN:  JMP     SHORT MAIN
; -------------------------------------------
DIVDND  DB      '3698'          ;Элементы данных
DIVSOR  DB      '4'
QUOTNT  DB      4 DUP(0)
; -------------------------------------------
MAIN    PROC    NEAR
        MOV     CX,04           ;4 цикла
        SUB     AH,AH           ;Стереть левый байт делимого
        AND     DIVSOR,0FH      ;Стереть ASCII 3 в делителе
        LEA     SI,DIVDND
        LEA     DI,QUOTNT
A20:
        MOV     AL,[SI]         ;Загрузить ASCII байт
                                ; (можно LODSB)
        AND     AL,0FH          ;Стереть ASCII тройку
        AAD                     ;Коррекция для деления
        DIV     DIVSOR          ;Деление
        MOV     [DI],AL         ;Сохранить частное
        INC     SI
        INC     DI
        LOOP    A20             ;Циклиться 4 раза
        RET
MAIN    ENDP
CODEGS  ENDS
        END     BEGIN
```

Рис.13.3. Деление в ASCII-формате.

ДВОИЧНО-ДЕСЯТИЧНЫЙ ФОРМАТ (BCD)

В предыдущем примере деления в ASCII-формате было получено частное 00090204. Если сжать это значение, сохраняя только правые цифры каждого байта, то получим 0924. Такой формат называется двоично-десятичным (BCD - Binary Coded Decimal) (или упакованным). Он содержит только десятичные цифры от 0 до 9. Длина двоично-десятичного представления в два раза меньше ASCII-представления.

Заметим, однако, что десятичное число 0924 имеет основание 10 и, будучи преобразованным в основание 16 (т.е. в шест. представление), даст шест.039C.

Можно выполнять сложение и вычитание чисел в двоично-десятичном представлении (BCD-формате). Для этих целей имеются две корректирующих команды:

DAA (Decimal Adjustment for Addition - десятичная коррекция для сложения)
DAS (Decimal Adjustment for Subtraction - десятичн. коррекция для вычит.)

Обработка полей также осуществляется по одному байту за одно выполнение. В примере программы, приведенном на рис.13.4, выполняется преобразование чисел из ASCII-формата в BCD-формат и сложение их. Процедура B10CONV преобразует ASCII в BCD. Обработка чисел может выполняться как справа налево, так и слева направо. Кроме того, обработка слов проще, чем обработка байтов, так как для генерации одного байта BCD-кода требуется два байта ASCII-кода. Ориентация на обработку слов требует четного количества байтов в ASCII-поле.

Процедура C10ADD выполняет сложение чисел в BCD-формате. Окончательный результат - 127263.

```
TITLE BCDADD (COM) Преобр.ASCII в BCD, сложение
CODESG  SEGMENT PARA "Code"
       ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
       ORG     100H
BEGIN:  JMP     SHORT MAIN
; -----------------------------------------
ASC1    DB      '057836'
ASC2    DB      '069427'
BCD1    DB      '000'
BCD2    DB      '000'
BCD3    DB      4 DUP(0)
; -----------------------------------------
MAIN    PROC    NEAR
       LEA     SI,ASC1+4          ;Инициализировать для ASC1
       LEA     DI,BCD1+2
       CALL    B10CONV            ;Вызвать преобразование
       LEA     SI,ASC2+4          ;Инициализировать для ASC2
       LEA     DI,BCD2+2
       CALL    B10CONV            ;Вызвать преобразование
       CALL    C10ADD             ;Вызвать сложение
       RET
MAIN    ENDP
;             Преобразование ASCII в BCD:
;             -------------------------
B10CONV PROC
       MOV     CL,04              ;Фактор сдвига
       MOV     OX,03              ;Число слов B20:
       MOV     AX,[SI]            ;Получить ASCII-пару
                           (можно использовать LODSW)
       XCHG    AH,AL
       SHL     AL,CL              ;Удалить тройки
       SHL     AX,CL              ; ASCII-кода
       MOV     [DI],AH            ;Записать BCD-цифру
       DEC     SI
       DEC     SI
       DEC     DI
       DEC     DX
       JNZ     B20
       RET
B10CONV ENDP
;             Сложение BCD-чисел:
```

```
;              ------------------
C10ADD PROC
      XOR      AH,AH             ;0чистить AH
      LEA      SI,BCD1+2          ;Инициализация
      LEA      DI,BCD2+2          ; BCD
      LEA      BX,BCD3+3          ; адресов
      MOV      CX,03             ;Трехбайтные поля
      CLC
C20:
      MOV      AL,[SI]           ;Получить BCD1 (или LODSB)
      ADC      AL,[DI]           ;Прибавить BCD2
      DAA                        ;Десятичная коррекция
      MOV      [BX],AL           ;Записать в BCD3
      DEC      SI
      DEC      DI
      DEC      BX
      LOOP     C20               ;Цикл 3 раза
      RET
C10ADD ENDP

CODESG ENDS
      END      BEGIN
```

Рис.13.4. BCD-преобразование и арифметика.

ПРЕОБРАЗОВАНИЕ ASCII-ФОРМАТА В ДВОИЧНЫЙ ФОРМАТ

Выполнение арифметических операций над числами в ASCII или BCD форматах удобно лишь для коротких полей. В большинстве случаев для арифметических операций используется преобразование в двоичный формат. Практически проще преобразование из ASCII-формата непосредственно в двоичный формат, чем преобразование из ASCII- в BCD-формат и, затем, в двоичный формат:

Метод преобразования базируется на том, что ASCII-формат имеет основание 10, а компьютер выполняет арифметические операции только над числами с основанием 2. Процедура преобразования заключается в следующем:

1. Начинают с самого правого байта числа в ASCII-формате и обрабатывают справа налево.
2. Удаляют тройки из левых шест.цифр каждого ASCII-байта.
3. Умножают ASCII-цифры на 1, 10, 100 (шест.1, A, 64) и т.д. и складывают результаты.

Для примера рассмотрим преобразование числа 1234 из ASCII-формата в двоичный формат:

	Десятичное	Шестнадцатиричное
4 x 1 =	4	4
3 x 10 =	30	1E
2 x 100 =	200	C8
1 x 1000 =	1000	3E8
Результат:		04D2

Проверьте, что шест.04D2 действительно соответствует десятичному 1234. На рис.13.5 в процедуре B10ASBI выполняется преобразование ASCII-числа 1234 в двоичный формат. В примере предполагается, что длина ASCII-числа равна 4 и она записана в поле ASCLEN. Для инициализации адрес

ASCII-поля ASCVAL-1 заносится в регистр SI, а длина - в регистр BX.
Команда по метке B20 пересылает ASCII-байт в регистр AL:

```
MOV  AL,[SI+BX]
```

Здесь используется адрес ASCVAL-1 плюс содержимое регистра BX (4), т.е. получается адрес ASCVAL+3 (самый правый байт поля ASCVAL). В каждом цикле содержимое регистра BX уменьшается на 1, что приводит к обращению к следующему слева байту. Для данной адресации можно использовать регистр BX, но не CX, и, следовательно, нельзя применять команду LOOP. В каждом цикле происходит также умножение поля MULT10 на 10, что дает в результате множители 1,10,100 и т.д. Такой прием применен для большей ясности, однако, для большей производительности множитель можно хранить в регистре SI или DI.

```
TITLE   EXCONV (COM) Преобр. ASCII и дв. ф-тов
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
        ORG     100H
BEGIN:  JMP     SHORT MAIN
; ------------------------------------------
ASCVAL  DB      '1234'          ;Элементы данных
BINVAL  DB      0
ASCLEN  DB      4
MULT10  DB      1
; ------------------------------------------
MAIN    PROC    NEAR            ;Основная процедура:
        CALL    B10ASBI         ;Вызвать преобразование ASCII
        CALL    C10BIAS         ;Вызвать преобразование двоичное
        RET
MAIN    ENDP
;               -------------------------------
;               Преобразование ASCII в двоичное:
;               -------------------------------
B10ASBI PROC
        MOV     CX,10           ;Фактор умножения
        LEA     SI,ASCVAL-1     ;Адрес ASCVAL
        MOV     BX,ASCLEN       ;Длина ASCVAL
B20:
        MOV     AL,[SI+BX]      ;Выбрать ASCII-символ
        AND     AX,000FH        ;Очистить зону тройки
        MUL     MULT10          ;Умножить на фактор 10
        ADD     BINVAL,AX       ;Прибавить к двоичному
        MOV     AX,MULT10       ;Вычислить следующий
        MUL     CX              ; фактор умножения
        MOV     MULT10,AX
        DEC     BX              ;Последн. ASCII-символ?
        JNZ     B20             ; Нет - продолжить
        RET
B10ASBI ENDP
;               -------------------
;               Преобр. дв. в ASCII:
;               -------------------
C10BIAS PROC
        MOV     CX,0010         ;Фактор деления
        LEA     SI,ASCVAL+3     ;Адрес ASCVAL
        MOV     AX,BINVAL       ;Загрузить дв. число
C20:
        CMP     AX,0010         ;Значение меньше 10?
```

```
       JB      C30              ; Да - выйти
       XOR     DX,DX             ;Очистить часть частного
       DIV     CX               ;Разделить на 10
       OR      DL,30H
       MOV     [SI],DL            ;Записать ASCII-символ
       OEC     SI
       JMP     C20
C30:
       OR      AL,30H            ;Записать посл. частное
       MOV     [SI],AL           ; как ASCII-символ
       RET
C10BIAS ENDP

CODESG  ENDS
       END    BEGIN
```

Рис.13.5. Преобразование ASCII и двоичного форматов.

ПРЕОБРАЗОВАНИЕ ДВОИЧНОГО ФОРМАТА В ASCII-ФОРМАТ

Для того, чтобы напечатать или отобразить на экране арифметический результат, необходимо преобразовать его в ASCII-формат. Данная операция включает в себя процесс обратный предыдущему. Вместо умножения используется деление двоичного числа на 10 (шест.0A) пока результат не будет меньше 10. Остатки, которые лежат в границах от 0 до 9, образуют число в ASCII-формате. В качестве примера рассмотрим преобразование шест.4D2 обратно в десятичный формат:

```
                  Частное      Остаток

         4D2 : A       7B           4
          7B : A       C           3
           C : A       1           2
```

Так как последнее частное 1 меньше, чем шест.A, то операция завершена. Остатки вместе с последним частным образуют результат в ASCII-формате, записываемый справа налево 1234. Все остатки и последнее частное должны записываться в память с тройками, т.е. 31323334.

На рис.13.5 процедура C10BIAS преобразует шест.4D2 (результат вычисления в процедуре B10ASBI) в ASCII-число 1234. Полезно переписать всю программу (рис.13.5) в компьютер и выполнить трассировку ее выполнения по шагам.

СДВИГ И ОКРУГЛЕНИЕ

Рассмотрим процесс округления числа до двух десятичных знаков после запятой. Если число равно 12,345, то необходимо прибавить 5 к отбрасываемому разряду и сдвинуть число вправо на один десятичный разряд:

```
         Число:            12,345
         Плюс 5:               +5
                          ------
         Округленное число:  12,350 = 12,35
```

Если округляемое число равно 12,3455, то необходимо прибавить 50 и сдвинуть на два десятичных разряда. Для 12,34555 необходимо прибавить 500

и сдвинуть на три десятичных разряда:

```
          12,3455          12,34555
             +50              +500
          -------         --------
          12,3505 = 12,35    12,35055 = 12,35
```

К числу, имеющему шесть знаков после запятой, необходимо прибавить 5000 и сдвинуть на четыре десятичных разряда и т.д. Поскольку данные представляются в компьютере в двоичном виде, то 12345 выглядит как шест.3039. Прибавляя 5 к 3039, получим 303E, что соответствует числу 12350 в десятичном представлении. Пока все хорошо. Но вот сдвиг на одну двоичную цифру дает в результате шест.181F, или 1675 - т.е. сдвиг на одну двоичную цифру просто делит число пополам. Но нам необходим такой сдвиг, который эквивалентен сдвигу вправо на одну десятичную цифру. Такой сдвиг можно осуществить делением на 10 (шест.A):

Шест.303E : Шест.A = 4D3 или дес.1235

Преобразование шест.4D3 в ASCII-формат дает число 1235. Теперь остается лишь вставить запятую в правильную позицию числа 12,35, и можно выдать на экран округленное и сдвинутое значение.

Таким образом можно округлять и сдвигать любые двоичные числа. Для трех знаков после запятой необходимо прибавить 5 и разделить на 10, для четырех знаков после запятой: прибавить 50 и разделить на 100. Возможно вы заметили модель: фактор округления (5, 50, 500 и т.д.) всегда составляет половину фактора сдвига (10, 100, 1000 и т.д.).

Конечно, десятичная запятая в двоичном числе только подразумевается.

ПРОГРАММА: ПРЕОБРАЗОВАНИЕ ВРЕМЕНИ И РАСЦЕНКИ РАБОТ ДЛЯ РАСЧЕТА ЗАРПЛАТЫ

Программа, приведенная на рис.13.6, позволяет вводить с клавиатуры значения продолжительности и расценки работ и отображать на экран расчитанную величину заработанной платы. Для краткости в программе опущены некоторые проверки на ошибку. Программа содержит следующие процедуры:

B10INPT Вводит значения времени работы на ее расценку с клавиатуры. Эти значения могут содержать десятичную запятую.

D10HOUR Выполняет преобразование значения времени из ASCII в двоичный формат.

E10RATE Выполняет преобразование значения расценки из ASCII в двоичный формат.

F10MULT Выполняет умножение, округление и сдвиг. Величина зарплаты без дробной части или с одним или двумя знаками после запятой не требует округления и сдвига. Данная процедура ограничена тем, что позволяет обрабатывать величину зарплаты с точностью до шести десятичных знаков, что, конечно, больше, чем требуется.

G10WAGE Вставляет десятичную запятую, определяет правую позицию для начала записи ASCII символов и преобразует двоичное значение зарплаты в ASCII-формат.

K10DISP Заменяет лидирующие нули на пробелы и выводит результат на экран

M10ASBI Преобразует ASCII в двоичный формат (общая процедура для времени и расценки) и определяет число цифр после запятой в введенном значении.

```
TITLE   SCREMP  (EXE) Ввод времени и расценки,
                  ;вывод величины оплаты
```

```
; --------------------------------------------------
STACKSG SEGMENT PARA STACK 'Stack'
        DW      32 DUP(?)
STACKSG ENDS
; --------------------------------------------------
DATASG  SEGMENT PARA 'Data'
HRSPAR  LABLE   BYTE            ;Список параметров для
                                ; ввода времени:
MAXHLEN DB      6               ;---------------------
ACTHLEN DB      ?
HRSFLD  DB      6 DUP(?)

RATEPAR LABLE   BYTE            ;Список параметров для
                                ; ввода расценки:
MAXRLEN DB      6               ;---------------------
ACTRLEN DB      ?
RATEFLN DB      6 DUP(?)

MESSG1  DB      'Hours worked? ','$'
MESSG2  DB      'Rate of pay? ','$'
MESSG3  DB      'Wage = '
ASCWAGE DB      10 DUP(30H), 13, 10, '$'
ADJUST  DW      ?
ASCHRS  DB      0
ASCRATE DB      0
BINVAL  DW      00
BINHRS  DW      00
BINRATE DW      00
COL     DB      00
DECIND  DB      00
MULT10  DW      01
NODEC   DW      00
ROW     DB      00
SHIFT   DW      ?
TENWD   DW      10
DATASG  ENDS
; --------------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
        ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASG
        PUSH    DS
        SUB     AX,AX
        PUSH    AX
        MOV     AX,DATASG
        MOV     DS,AX
        MOV     ES,AX
        MOV     AX,0600H
        CALL    Q10SCR          ;Очистить экран
        CALL    Q20CURS         ;Установить курсор
A20LOOP:
        CALL    B10INPT         ;Ввести время и расценку
        CMP     ACTHLEN,00      ;Завершить работу?
        JE      A30
        CALL    D10HOUR         ;Получить двоичное время
        CALL    E10RATE         ;Получить двоичную расценку
        CALL    F10MULT         ;Расчитать оплату
        CALL    G10WAGE         ;Преобразовать в ASCII
        CALL    K10DISP         ;Выдать результат на экран
        JMP     A20LOOP
A30:
        MOV     AX,0600H
```

```
       CALL    Q10SCR          ;Очистить экран
       RET                    ;Выйти из программы
BEGIN   ENDP
;              Ввод времени и расценки
; ------------------------------------------------
B10INPT PROC
       LEA     DX,MESSG1       ;Запрос для ввода времени
       MOV      AH,09
       INT     21H
       LEA     DX,HRSPAR       ;Ввести время
       MOV      AH,0AH
       INT     21H
       CMP     ACTHLEN,00      ;Пустой ввод?
       JNE     B20
       RET                    ; да - вернуться A20LOOP
B20:
       MOV      COL,25         ;Установить столбец
       CALL    Q20CURS
       LEA     DX,MESSG2       ;Запрос для ввода расценки
       MOV      AH,09
       INT     21H
       LEA     DX,RATEPAR      ;Ввести расценку
       MOV      AH,0AH
       INT     21H
       RET
B10INPT ENDP
;              Обработка времени:
;              -----------------
D10HOUR PROC
       MOV      NODEC,00
       MOV      CL,ACTHLEN
       SUB     CH,CH
       LEA     SI,HRSFLD-1     ;Установить правую позицию
       ADD      SI,CX          ; времени
       CALL    M10ASBI         ;Преобразовать в двоичное
       MOV      AX,BINVAL
       MOV      BINHRS,AX
       RET
D10HOUR ENDP
;              Обработка расценки:
;              ------------------
E10RATE PROC
       MOV      CL,ACTRLEN
       SUB     CH,CH
       LEA     SI,RATEFLD-1    ;Установить правую позицию
       ADD      SI.CX          ; расценки
       CALL    M10ASBI         ;Преобразовать в двоичное
       MOV      AX,BINVAL
       MOV      BINRATE,AX
       RET
E10RATE ENDP
;              Умножение, округление и сдвиг:
;              ----------------------------
F10MULT PROC
       MOV      CX,05
       LEA     DI,ASCWAGE      ;Установить формат оплаты
       MOV      AX,3030H       ; в код ASCII (30)
       CLD
       REP STOSW
       MOV      SHIFT,10
       MOV      ADJUST,00
```

```
       MOV     CX,NODEC
       CMP     CL,06           ;Если более 6 десятичных
       JA      F40             ; знаков, то ошибка
       DEC     CX
       DEC     CX
       JLE     F30             ;Обойти, если менее 3 знаков
       MOV     NODEC,02
       MOV     AX,01
F20:
       MUL     TENWD           ;Вычислить фактор сдвига
       LOOP    F20
       MOV     SHIFT,AX
       SHR     AX,1            ;Округлить результат
       MOV     ADJUST,AX
F30:
       MOV     AX,BINHRS
       MUL     BINRATE         ;Вычислить оплату
       ADD     AX,ADJUST       ;Округлить оплату
       ADC     DX,00
       CMP     DX,SHIFT        ;Результат слишком велик
       JB      F50             ; для команды DIV?
F40:
       SUB     AX,AX
       JMP     F70
F50:
       CMP     ADJUST,00       ;Сдвиг нее требуется?
       JZ      F80
       DIV     SHIFT           ;Сдвинуть оплату
F70:   SUB     DX,DX           ;Стереть остаток
F80:   RET
F10MULT ENDP
;            Преобразование в ASCII формат:
;            -----------------------------
G10WAGE PROC
       LEA     SI,ASCWAGE+7    ;Установить дес. точку
       MOV     BYTE PTR[SI],'.'
       ADD     SI,NODEC        ;Установить правую позицию
G30:
       CMP     BYTE PTR[SI],'.'
       JNE     G35             ;Обойти, если дес.поз.
       DEC     SI
G35:
       CMP     DX,00           ;Если dx:ax < 10,
       JNZ     G40
       CMP     AX,0010         ; то операция завершена
       JB      G50
G40:
       DIV     TENWD           ;Остаток - ASCII-цифра
       OR      DL,30H
       MOV     [SI],DL         ;Записать ASCII символ
       DEC     SI
       SUB     DX,DX           ;Стереть остаток
       JMP     G30
G50:
       OR      AL,30H          ;Записать последний ASCII
       MOV     [SI],AL         ; символ
       RET
G10WAGE ENDP
;            Вывод величины оплаты:
;            ---------------------
K10DISP PROC
```

```
        MOV     COL,50          ;Установить столбец
        CALL    Q20CURS
        MOV     CX,09
        LEA     SI,ASCWAGE
K20:                            ;Стереть лидирующие нули
        CMP     BYTE PTR[SI],30H
        JNE     K30             ; пробелами
        MOV     BYTE PTR[SI],20H
        INC     SI
        LOOP    K20
K30:
        LEA     DX,MESSG3       ;Вывод на экран
        MOV     AH,09
        INT     21H
        CMP     ROW,20          ;Последняя строка экрана?
        JAE     K80
        INC     ROW             ; нет - увеличить строку
        JMP     K90
K80:
        MOV     AX,0601H        ; да --
        CALL    Q10SCR          ; прокрутить и
        MOV     COL,00          ; установить курсор
        CALL    Q20CURS
K90:    RET
K10DISP ENDP
;               Преобразование ASCII-чисел
;               в двоичное представление:
;               --------------------------
M10ASBI PROC
        MOV     MULT10,0001
        MOV     BINVAL,00
        MOV     DECIND,00
        SUB     BX,BX
M20:
        MOV     AL,[SI]         ;ASCII-символ
        CMP     AL,'.'          ;Обойти, если дес.точка
        JNE     M40
        MOV     DECIND,01
        JMP     M90
M40:
        AND     AX,000FH
        MUL     MULT10          ;Умножить на фактор
        ADD     BINVAL,AX       ;Сложить с дв.значением
        MOV     AX,MULT10       ;Вычислить следующий
        MUL     TENVD           ; фактор x 10
        MOV     MULT10,AX
        CMP     DECIND,00       ;Десятичная точка?
        JNZ     M90
        INC     BX              ; да - обойти точку
M90:
        DEC     SI
        LOOP    M20
                                ;Конец цикла
        CMP     DECIND,00       ;Была дес.точка?
        JZ      M100            ; да --
        ADD     NODEC,BX        ; сложить с итогом
M100:   RET
M10ASBI ENDP
;               Прокрутка экрана:
;               ----------------
Q10SCR  PROC    NEAR            ;AX установлен при вызове
```

```
       MOV     BH,30           ;Цвет (07 для ч/б)
       SUB     CX,CX
       MOV     DX,184FH
       INT     10H
       RET
Q10SCR  ENDP
;              Установка курсора:
;              -----------------
Q20CURS PROC    NEAR
       MOV     AH,02
       SUB     BH,BH
       MOV     DH,ROW
       MOV     DL,COL
       INT     10H
       RET
Q20CURS ENDP

CODESG  ENDS
       END     BEGIN
```

Рис.13.6. Расчет заработной платы.

О г р а н и ч е н и я. Первое ограничение в программе, приведенной на рис.13.6, состоит в том, что допускает не более шести десятичных знаков после запятой. Другое ограничение - размер самой зарплаты и тот факт, что сдвиг включает деление на число, кратное 10, а преобразование в ASCII-формат включает деление на 10. Если значение времени или расценки содержит больше шести десятичных знаков или зарплата превышает величину около 655350, то программа выдает нулевой результат. На практике программа может предусмотреть в данном случае вывод предупреждающего сообщения или иметь подпрограммы для исключения таких ограничений.

К о н т р о л ь о ш и б о к. Программа, разработанная для пользователей, не являющихся программистами, должна не только выдавать предупреждающие сообщения, но также проверять корректность вводимых значений. Правильными символами при вводе числовых значений являются цифры от 0 до 9 и символ десятичной запятой. Для любых других символов программа должна выдать предупреждающее сообщение и вновь повторить запрос на ввод. Полезной командой для проверки корректности вводимых символов является XLAT (см. гл.14).

Тщательно проверяйте программы для любых возможных состояний: нулевое значение, максимально большие и малые значения, отрицательные значения.

Отрицательные величины

Некоторые применения программ допускают наличие отрицательных величин. Знак минус может устанавливаться после числа, например, 12,34-, или перед числом -12,34. Программа может проверять наличие минуса при преобразовании в двоичный формат. Можно оставить двоичное число положительным, но установить соответствующий индикатор исходной отрицательной величины. После завершения арифметических операций знак минус при необходимости может быть вставлен в ASCII поле.

Если необходимо, чтобы двоичное число было также отрицательным, то можно преобразовать, как обычно, ASCII-формат в двоичный, а для изменения знака двоичного числа воспользоваться командами, описанными в гл.12 "Преобразование знака". Будьте внимательны при использовании команд IMUL и IDIV для обработки знаковых данных. Для округления отрицательных чисел следует не прибавлять, а вычитать фактор 5.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- ASCII-формат требует один байт на каждый символ. Если поле содержит только цифры от 0 до 9, то замена старших троек в каждом байте на нули создает распакованный десятичный формат. Сжатие числа до двух цифр в байте создает упакованный десятичный формат.

- После ASCII-сложения необходимо выполнить коррекцию с помощью команды AAA; после ASCII-вычитания - коррекция с помощью команды AAS.

- Прежде чем выполнить ASCII-умножение, необходимо преобразовать множимое и множитель в "распакованный десятичный" формат, обнулив в каждом байте левые тройки. После умножения необходимо выполнить коррекцию результата с помощью команды AAM.

- Прежде чем выполнить ASCII-деление, необходимо: 1) преобразовать делимое и делитель в "распакованный десятичный" формат, обнулив в каждом байте левые тройки и 2) выполнить коррекцию делимого с помощью команды AAD.

- Для большинства арифметических операций используйте преобразование чисел из ASCII-формата в двоичной формат. В процессе такого преобразования проверяйте на корректность ASCII-символы: они должны быть от шест.30 до шест.39, могут содержать десятичную запятую (точку) и, возможно, знак минус.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

13.1. Предположим, что регистр AX содержит 9 в ASCII коде, а регистр BX -7 также в ASCII коде. Объясните и дайте точный результат для следующих несвязанных операций:

```
а)  ADD  AX,33H         б)  ADD  AX,BX
    AAA                     AAA
в)  SUB  AX,BX          г)  SUB  AX,0DH
    AAS                     AAS
```

13.2. Поле UNPAK содержит шест. 01040705 в распаковочном десятичном формате. Напишите цикл, который преобразует это содержимое в ASCII-формат, т.е. 31343735.

13.3. Поле ASCA содержит значение 313733 в ASCII-формате, а другое поле ASCB содержит 35. Напишите команды для умножения этих чисел в ASCII-формате и записи произведения в поле ASCPRO.

13.4. Используя данные из вопроса 13.3, разделите ASCA на ASCB и запишите частное в поле ASCQUO.

13.5. Выполните следующие вычисления вручную: а) преобразовать ASCII 46328 в двоичный формат и показать результат в шест.виде; б) преобразовать полученное шест. значение обратно в ASCII-формат.

13.6. Напишите и выполните программу, которая определяет размер памяти компьютера (INT 12H - см. гл.2), преобразует полученное значение в ASCII-формат и выводит результат на экран в следующем виде:

Размер памяти nnn байтов.

ГЛАВА 14 Обработка таблиц

Ц е л ь: Раскрыть требования для определения таблиц, организации поиска в таблицах и сортировки элементов таблицы.

ВВЕДЕНИЕ

Многие программные применения используют табличную организацию таких данных, как имена, описания, размеры, цены. Определение и использование таблиц включает одну новую команду ассемблера - XLAT. Таким образом, использование таблиц - это лишь дело техники и применения знаний, полученных из предыдущих глав.

Данная глава начинается определением некоторых общепринятых таблиц. Организация поиска в таблице зависит от способа ее определения. Существует много различных вариантов определения таблиц и алгоритмов поиска.

ОПРЕДЕЛЕНИЕ ТАБЛИЦ

Для облегчения табличного поиска большинство таблиц определяются систематично, т.е. элементы таблицы имеют одинаковый формат (символьный или числовой), одинаковую длину и восходящую или нисходящую последовательность элементов.

Таблица, которой уже приходилось пользоваться в данной книге - это стек, представляющий собой таблицу из 64-х неинициализированных слов:

```
STACK     DW   64 DUP(?)
```

Следующие две таблицы инициализированы символьными и числовыми значениями:

```
MONTAB    DB   'JAN','FEB','MAR', ... ,'DEC'
COSTAB    DB   205,208,209,212,215,224,...
```

Таблица MONTAB определяет алфавитные аббревиатуры месяцев, а COSTAB - определяет таблицу номеров служащих. Таблица может также содержать смешанные данные (регулярно чередующиеся числовые и символьные поля). В следующей ассортиментной таблице каждый числовой элемент (инвентарный номер) имеет две цифры (один байт), а каждый символьный элемент (наименование) имеет девять байтов. Точки, показанные в наименовании "Paper" дополняют длину этого поля до 9 байт. Точки показывают, что недостающее пространство должно присутствовать. Вводить точки необязательно.

```
STOKTBL DB 12,'Computers',14,'Paper....',17,'Diskettes'
```

Для ясности можно закодировать элементы таблицы вертикально:

```
STOKTBL   DB   12,  'Computers'
          DB   14,  'Paper....'
          DB   17,  'Diskettes'
```

Рассмотрим теперь различные способы использования таблиц в

программах.

ПРЯМОЙ ТАБЛИЧНЫЙ ДОСТУП

Предположим, что пользователь ввел номер месяца - 03 и программа должна преобразовать этот номер в алфавитное значение March. Программа для выполнения такого преобразования включает определение таблицы алфавитных названий месяцев, имеющих одинаковую длину. Так как самое длинное название - September, то таблица имеет следующий вид:

```
MONTBL    DB   'January..'
          DB   'February.'
          DB   'March....'
```

Каждый элемент таблицы имеет длину 9 байт. Адрес элемента 'January' - MONTBL+0, 'February' - MONTBL+9, 'March' - MONTBL+18. Для локализации месяца 03, программа должна выполнить следующее:

1. Преобразовать введенный номер месяца из ASCII 33 в двоичное 03.
2. Вычесть единицу из номера месяца: 03 - 1 = 02
3. Умножить результат на длину элемента (9): 02 x 9 = 18
4. Прибавить произведение (18) к адресу MONTBL; в результате получится адрес требуемого названия месяца: MONTBL+18.

```
      page    60,132
TITLE   DIRECT  (COM) Прямой табличный доступ
CODESG  SEGMENT PARA 'Code'
      ASSUME  CS:CODESG,DS:CODESG,ES:CODESG
      ORG     100H
BEGIN:  JMP     SHORT MAIN
; ----------------------------------------------
THREE   DB      3
MONIN   DB      '11'
ALFMON  DB      '???','$'
MONTAB  DB      'JAN','FEB','MAR','APR','MAY','JUN'
      DB      'JUL','AUG','SEP','OKT','NOV','DEC'
; ----------------------------------------------
MAIN    PROC    NEAR           ;Основная процедура
      CALL    C10CONV        ;Получить двоичное значение
      CALL    D10LOC         ;Выделить месяц из таблицы
      CALL    F10DISP        ;Выдать месяц на экран
      RET
MAIN    ENDP
;             Перевод ASCII в двоичное представление:
;             -------------------------------------
C10CONV PROC
      MOV     AH,MONIN       ;Загрузить номер месяца
      MOV     AL,MONIN+1
      XOR     AX,3030H       ;Удалить ASCII тройки
      CMP     AH,00          ;Месяц 01-09?
      JZ      C20            ; да - обойти
      SUB     AH,AH          ; нет - очистить AH,
      ADD     AL,10          ; и перевести в двоичное
C20     RET
C10CONV ENDP
;             Выделение месяца из таблицы:
```

```
;             --------------------------
D10LOC  PROC
       LEA     SI,MONTAB
       DEC     AL             ;Коррекция для таблицы
       MUL     THREE           ;Умножить AL на 3
       ADD     SI,AX
       MOV     CX,03          ;Трехсимвольная пересылка
       CLD
       LEA     DI,ALFMON
       REP MOVSB               ;Переслать 3 символа
       RET
D10LOC  ENDP
;             Вывод на экран симв.месяца:
;             -------------------------
F10DISP PROC
       LEA     DX,ALFMON
       MOV     AH,09
       INT     21H
       RET
F10DISP ENDP
CODESG  ENDS
       END     BEGIN
```

Рис.14.1. Прямая табличная адресация.

На рис.14.1 приведен пример прямого доступа к таблице названий месяцев. Для краткости в программе используются вместо девятисимвольных названий - трехсимвольные. Введенный номер месяца определен в поле MONIN. Предположим, что некоторая подпрограмма формирует запрос на ввод номера месяца в ASCII-формате в поле MONIN.

Описанная техника работы с таблицей называется прямым табличным доступом. Поскольку данный алгоритм непосредственно вычисляет адрес необходимого элемента в таблице, то в программе не требуется выполнять операции поиска.

Хотя прямая табличная адресация очень эффективна, она возможна только при последовательной организации. То есть можно использовать такие таблицы, если элементы располагаются в регулярной последовательности: 1, 2, 3,... или 106, 107, 108,... или даже 5, 10, 15. Однако, не всегда таблицы построены таким образом. В следующем разделе рассматриваются таблицы, имеющие нерегулярную организацию.

ТАБЛИЧНЫЙ ПОИСК

Некоторые таблицы состоят из чисел, не имеющих видимой закономерности. Характерный пример - таблица инвентарных номеров с последовательными номерами, например, 134, 138, 141, 239 и 245. Другой тип таблиц состоит из распределенных по ранжиру величин, таких как подоходный налог. В следующих разделах рассмотрим эти типы таблиц и организацию табличного поиска.

Таблицы с уникальными элементами

Инвентарные номера большинства фирм часто не имеют последовательного порядка. Номера, обычно, группируются по категориям, первые цифры указывают на мебель или приборы, или номер отдела. Кроме того время от времени номера удаляются, а новые добавляются. В таблице необходимо

связать инвентарные номера и их конкретные наименования (и, если требуется, включить стоимость). Инвентарные номера и наименования могут быть определены в различных таблицах, например:

```
STOKNOS DB '101','107','109',...
STOKDCR DB 'Excavators','Processors','Assemblers',...
```

или в одной таблице, например:

```
STOKTAB   DB   '101','Excavators'
          DB   '107','Processors'
          DB   '109','Assemblers'
          ...
```

Программа на рис.14.2 определяет инвентарную таблицу и выполняет табличный поиск. Таблица содержит шесть пар номеров и наименований. Цикл поиска начинается со сравнения введенного инвентарного номера в поле STOKNIN с первым номером в таблице. Если номера различные, то адрес в таблице увеличивается для сравнения со следующим инвентарным номером. Если номера равны, то программа (A30) выделяет наименование из таблицы и записывает его в поле DESCRN.

Поиск выполняет максимум шесть сравнений и если требуемый номер в таблице отсутствует, то происходит переход на программу обработки ошибки, которая выводит на экран соответствующее сообщение.

Обратите внимание, что в начале программы имеется команда, которая пересылает содержимое поля STOKNIN в регистр AX. Хотя STOKNIN определенно как 3233, команда MOV загрузит в регистр AX это значение в обратной последовательности байтов 3332. Так как элементы таблицы имеют прямую последовательность байтов, то после команды MOV имеется команда XCHG, которая меняет местами байты в регистре AX, возвращая им прямую последовательность, т.е. 3233. Команда CMP, предполагая обратную последовательность, сравнивает сначала правые байты, а затем - левые. Следовательно, проверка на равенство будет корректной, но проверки на больше или меньше дадут неправильные результаты. Для сравнения на больше или меньше следует опустить команду XCHG, переслать элемент таблицы командой MOV, скажем, в регистр BX и затем сравнить содержимое регистров AX и BX следующим образом:

```
    MOV  AX,STOKNIN
    LEA  SI,STOKTAB
C20:
    MOV  BX,[SI]
    CMP  AX,BX
    JA или JB ...
```

В программе такого типа другая таблица может определять стоимость единицы товара. Программа может локализовать элемент таблицы, вычислить продажную стоимость (количество товара умножить на стоимость единицы товара) и выдать на экран наименование и продажную стоимость товара.

В примере на рис.14.2 таблица содержит двухбайтовые номера и десятибайтовые наименования. Детальное программирование будет отличаться для различного числа и длины элементов. Например, для сравнения трехбайтовых полей можно использовать команду REPE CMPSB, хотя эта команда также включает использование регистра CX.

```
      page    60,132
TITLE   TABSRCH (COM) Табличный поиск
CODESG  SEGMENT PARA 'Code'
      ASSUME  CS:CODESG,DS:CODESG,ES:CODESG
```

```
      ORG     100H
BEGIN:  JMP     SHORT MAIN
; ---------------------------------------------
STOKNIN DW      '23'
STOKTAB DB      '05','Excavators'
      DB      '08','Lifters   '
      DB      '09','Presses   '
      DB      '12','Valves    '
      DB      '23','Processors'
      DB      '27','Pumps     '
DESCRN  10      DUP(?)
; ---------------------------------------------
MAIN    PROC    NEAR
      MOV     AX,STOKNIN      ;Загрузить номер элемента
      XCHG    AL,AH
      MOV     CX,06           ;Число элементов в таблице
      LEA     SI,STOKTAB      ;Начальный адрес таблицы
A20:
      CMP     AX,[SI]         ;Сравнить элементы
      JE      A30             ;Если равны - выйти,
      ADD     SI,12           ; нет - следующий элемент
      LOOP    A20
      CALL    R10ERR          ;Элемент в таблице не найден
      RET
A30:
      MOV     CX,05           ;Длина описания элемента
      LEA     DI,DESCRN       ;Адрес описания элемента
      INC     SI
      INC     SI              ;Выделить описание
      REP MOVSW               ; из таблицы
      RET
MAIN    ENDP
;
R10ERR  PROC
;           <Вывод сообщения об ошибке>
      RET
R10ERR  ENDP

CODESG  ENDS
      END     BEGIN
```

Рис.14.2. Табличный поиск

Таблицы с ранжированием

Подоходный налог дает характерный пример таблицы с ранжированными значениями. Представим себе таблицу, содержащую размеры доходов облагаемых налогами, процент налога и поправочный коэффициент:

Размер дохода	Процент налога	Поправочный к-нт
0-1000.00	10	0,00
1000,01-2500,00	15	050,00
2500,01-4250,00	18	125,00
4250,01-6000,00	20	260,00
6000,01 и более	23	390,00

В налоговой таблице процент увеличивается в соответствии с увеличением налогооблагаемого дохода. Элементы таблицы доходов содержат максимальные

величины для каждого шага:

```
TAXTBL    DD   100000,250000,425000,600000,999999
```

для организации поиска в такой таблице, программа сравнивает доход налогоплательщика с табличным значением дохода:

- если меньше или равно, то использовать соответствующий процент и поправку;
- если больше, то перейти к следующему элементу таблицы.

Величина налога рассчитывается по формуле:

Доход x Процент налога : 100 - поправочный к-нт

Табличный поиск с использованием сравнения строк

Если элемент таблицы превышает длину в два байта, то для операции сравнения можно использовать команду REPE CMPS. Предположим, что таблица инвентарных номеров (рис.14.2) переделана для трехбайтовых номеров. Если STOKNIN является первым полем в области данных, а STOKTAB - вторым, то они могут выглядеть следующим образом:

```
Данные: |123|035Excavators|038Lifters   |049Presses   | ...
        |   |  |           |  |          |  |
Адрес:  00  03 06          16 19         29 32
```

Программа на рис.14.3 определяет таблицу STOKTAB, включая последний элемент '999' для индикации конца таблицы при поиске. Программа поиска сравнивает содержимое каждого элемента таблицы с содержимым поля STOKNIN:

```
Элемент таблицы    STOKNIN   Результат сравнения

    035          123      Меньше: проверить след.эл-т
    038          123      Меньше: проверить след.эл-т
    049          123      Меньше: проверить след.эл-т
    102          123      Меньше: проверить след.эл-т
    123          123      Равно: элемент найден
```

Заметим, что команда CMPSB на рис.14.3 сравнивает байт за байтом, пока байты не будут равны и автоматически увеличивает регистры SI и DI.

Регистр CX инициализируется значением 03, а начальные относительные адреса в регистрах SI и DI устанавливаются равными 03 и 00 соответственно. Сравнение с первым элементом таблицы (035:123) завершается на первом байте, после этого регистр SI содержит 04, DI: 01, CX: 02. Для следующего сравнения регистр SI должен иметь значение 16, а DI: 00. Корректировка регистра DI сводится к простой перезагрузке адреса STOKNIN. Увеличение адреса следующего элемента таблицы, который должен быть в регистре SI, зависит от того, на каком байте (первом, втором или третьем) закончилось предыдущее сравнение. Регистр CX содержит число байт, не участвующих в сравнении, в данном случае - 02. Прибавив к содержимому регистра SI значение в регистре CX и длину наименования, получим относительный адрес следующего элемента:

```
Адрес в SI после CMPSB            04
Прибавить CX                      02
Прибавить длину наименования      10
                                  --
Относительный адрес след.элемента 16
```

Так как регистр CX всегда содержит число байт, не участвующих в сравнении (если такие есть), то расчет справедлив для всех случаев: прекращение сравнения после 1, 2 или 3 байта. Если сравниваются одинаковые элементы, то регистр CX получит значение 00, а адрес в регистре SI укажет на требуемое наименование.

```
      page    60,132
TITLE   TABSRCH (COM) Табличный поиск,  использующий CMPSB
CODESG  SEGMENT PARA 'Code'
      ASSUME  CS:CODESG,DS:CODESG,ES:CODESG
      ORG     100H
BEGIN:  JMP     SHORT MAIN
; ---------------------------------------------------
STOKNIN DW      '123'
STOKTAB DB      '035','Excavators'       ;Начало таблицы
      DB      '038','Lifters   '
      DB      '049','Presses   '
      DB      '102','Valves    '
      DB      '123','Processors'
      DB      '127','Pumps     '
      DB      '999', 10 DUP(' ')       ;Конец таблицы
DESCRN  10      DUP(?)
; ---------------------------------------------------
MAIN    PROC    NEAR
      CLD
      LEA     SI,STOKTAB     ;Начальный адрес таблицы
A20:
      MOV     CX,03          ;Сравнивать по 3 байта
      LEA     DI,STOKNIN     ;Адрес искомого элемента
      REPE CMPSB             ;Сравнение
      JE      A30            ;Если равно - выйти,
      JA      A40            ;если больше - нет в таблице
      ADD     SI,CX          ;Прибавить CX к адресу
      JMP     A20            ;Следующий элемент таблицы
A30:
      MOV     CX,05          ;Пересылать 5 слов
      LEA     DI,DESCRN      ;Адрес описания
      REP MOVSV              ;Переслать из таблицы
      RET
A40:
      CALL    R10ERR         ;элемент в таблице не найден
      RET
MAIN    ENDP

R10ERR  PROC
;           <Вывод на экран сообщения об ошибке>
      RET
R10ERR  ENDP

CODESG  ENDS
      END     BEGIN
```

Рис.14.3. Табличный поиск с использованием команды CMPSB

Таблицы с элементами переменной длины

Существуют таблицы, в которых элементы имеют переменную длину. Каждый

элемент такой таблицы может завершаться специальным символом ограничителем, например, шест.00; конец таблицы можно обозначить ограничителем шест.FF. В этом случае необходимо гарантировать, чтобы внутри элементов таблицы не встречались указанные ограничители. Помните, что двоичные числа могут выражаться любыми битовыми комбинациями. Для поиска можно использовать команду SCAS.

ТРАНСЛИРУЮЩАЯ КОМАНДА XLAT

Команда XLAT транслирует содержимое одного байта в другое предопределенное значение. С помощью команды XLAT можно проверить корректность содержимого элементов данных. При передаче данных между персональным компьютером и ЕС ЭВМ (IBM) с помощью команды XLAT можно выполнить перекодировку данных между форматами ASCII и EBCDIC.

В следующем примере происходит преобразование цифр от 0 до 9 из кода ASCII в код EBCDIC. Так как представление цифр в ASCII выглядит как шест.30-39, а в EBCDIC - шест.F0-F9, то замену можно выполнить командой OR. Однако, дополнительно преобразуем все остальные коды ASCII в пробел (шест.40) в коде EBCDIC. Для команды XLAT необходимо определить таблицу перекодировки, которая учитывает все 256 возможных символов, с кодами EBCDIC в ASCII позициях:

```
XLTBL  DB   47 DUP(40H)     ;Пробелы в коде EBCDIC
       DB   0F0H,0F1H,0F2H,0F3H,...,0F9H  ;0-9 (EBCDIC)
       DB   199 DUP(40H)    ;Пробелы в коде EBCDIC
```

Команда XLAT предполагает адрес таблицы в регистре BX, а транслируемый байт (например, поля ASCNO) в регистре AL. Следующие команды выполняют подготовку и трансляцию байта:

```
LEA  BX,XLTBL
MOV  AL,ASCNO
XLAT
```

Команда XLAT использует значение в регистре AL в качестве относительного адреса в таблице, т.е. складывает адрес в BX и смещение в AL. Если, например, ASCNO содержит 00, то адрес байта в таблице будет XLTBL+00 и команда XLAT заменит 00 на шест.40 из таблицы. Если поле ASCNO содержит шест.32, то адрес соответствующего байта в таблице будет XLTBL+50. Этот байт содержит шест.F2 (2 в коде EBCDIC), который команда XLAT загружает в регистр AL.

В программе на рис.14.4 добавлено преобразование десятичной точки (2E) и знака минус (2D) из кода ASCII в код EBCDIC (4B и 60 соответственно). В программе организован цикл для обработки шестибайтового поля. Поле ASCNO в начале выполнения программы содержит значение 31.5 с последующим пробелом, или шест.2D33312E3520. В конце выполнения программы в поле EBCNO должно быть шест.60F3F14BF540.

```
      page    60,132
TITLE   XLATE   (COM) Перевод кода ASCII в код EBCDIC
CODESG  SEGMENT PARA 'Code'
      ASSUME  CS:CODESG,DS:CODESG,ES:CODESG
      ORG     100H
BEGIN:  JMP     MAIN
; ------------------------------------------------
ASCNO   DB      '-31.5'
EBCNO   DB      6 DUP(' ')
```

```
XLTAB   DB      45 DUP(40H)
        DB      60H, 2DH
        DB      5CH
        DB      0F0H,0F1H,0F2H,0F3H,0F4H
        DB      0F5H,0F6H,0F7H,0F8H,0F9H
        DB      199 DUP(40H)
; ----------------------------------------------------
MAIN    PROC    NEAR            ;Основная процедура
        LEA     SI,ASCNO        ;Адрес символов ASCNO
        LEA     DI,EBCNO        ;Адрес поля EBCNO
        MOV     CX,06           ;Длина
        LEA     BX,XLTAB        ;Адрес таблицы
A20:
        MOV     AL,[SI]         ;Получить ASCII символ
        XLAT                    ;Перекодировка
        MOV     [DI],AL         ;Записать в поле EBCNO
        INC     DI
        INC     SI
        LOOP    A20             ;Повторить 6 раз
        RET
MAIN    ENDP
CODESG  ENDS
        END     BEGIN
```

Рис.14.4. Преобразование ASCII в EBCDIC.

ПРОГРАММА: ОТОБРАЖЕНИЕ ШЕСТ. И ASCII-КОДОВ

```
        page    60,132
TITLE   ASCHEX  (COM) Преобразование ASCII в шест.
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,ES:CODESG
        ORG     100H
BEGIN:  JMP     MAIN
; ----------------------------------------------
DISPROW DB      16 DUP('     '), 13
HEXSTR  DB      00
XLATAB  DB      30H,31H,32H,33H,34H,35H,36H,37H,38H,39H
        DB      41H,42H,43H,44H,45H,46H
; ----------------------------------------------
MAIN    PROC    NEAR            ;Основная процедура
        CALL    Q10CLR          ;Очистить экран
        LEA     SI,DISPROW
A20LOOP:
        CALL    C10HEX          ;Перекодировать
        CALL    D10DISP         ; и вывести на экран
        CMP     HEXCTR,0FFH     ;Последнее значение (FF)?
        JE      A50             ; да - завершить
        INC     HEXCTR          ; нет - перейти к следующему
        JMP     A20LOOP
A50:    RET
MAIN    ENDP

C10HEX  PROC    NEAR            ;Перекодировка в шест.
        MOV     AH,00
        MOV     AL,HEXCTR       ;Получить шест.пару
```

```
       SHR     AX,CL          ;Сдвиг правой шест.цифры
       LEA     BX,XLATAB       ;Установить адрес таблицы
       MOV     CL,04          ;Установить величину сдвига
       XLAT                  ;Перекодировка в шест.
       MOV     [SI],AL        ;Записать левый символ

       MOV     AL,HEXCTR
       SHL     AX,CL          ;Сдвиг левой цифры
       XLAT
       MOV     [SI]+1,AL      ;Перекодировка в шест.
       RET                   ;Записать правый символ
C10HEX  ENDP

D10DISP PROC    NEAR           ;Вывод на экран
       MOV     AL,HEXCTR
       MOV     [SI]+3,AL
       CMP     AL,1AH         ;Символ EOF?
       JE      D20            ; да - обойти
       CMP     AL,07H         ;Меньше/равно 08?
       JB      D30            ; да - OK
       CMP     AL,10H         ;Больше/равно 0F?
       JAE     D30            ; да - OK
D20:
       MOV     BYTE PTR [SI]+3,20H
D30:
       ADD     SI,05          ;Следующий элемент в строке
       LEA     DI,DISPROW+80
       CMP     DI,SI
       JNE     D40
       MOV     AH,40H         ;Функция вывода на экран
       MOV     BX,01          ;Номер устройства
       MOV     CX,81          ;Вся строка
       LEA     DX,DISPROW
       INT     21H
       LEA     SI,DISPROW     ;Начальный адрес строки
D40:   RET
D10DISP ENDP

Q10CLR  PROC    NEAR           ;Очистка экрана
       MOV     AX,0600H
       MOV     BH,03          ;Цвет (07 для ч/б)
       MOV     CX,0000
       MOV     DX,184FH
       INT     10H
       RET
Q10CLR  ENDP

CODESG  ENDS
       END     BEGIN
```

Рис.14.5. Отображение шест. и ASCII-кодов

Программа, приведенная на рис.14.5, отображает на экране почти все ASCII-символы, а также их шест. значения. Например, ASCII-символ для шест.53 - это буква S, эти данные программа выводит в виде 53 S. Полное изображение на экране выглядит в виде матрицы 16x16:

```
            00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
            .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

```
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```

Как было показано еще на рис.8.1, отображение ASCII-символов, особых проблем не вызывает. Что же касается отображения шест. значений в символах ASCII, то этот процесс более сложный. Например, для вывода на экран в коде ASCII шест.00, 01 и т.д. необходимо преобразовать шест.00 в шест.3030, шест.01 в шест.3031 и т.д.

В программе начальное значение поля HEXCTR равно 00. Это значение последовательно увеличивается на 1. Процедура C10HEX расщепляет байт HEXCTR на две щест. цифры. Предположим, что байт HEXCTR содержит шест.4F. Процедура сначала выделяет шест. цифру 4 и использует это значение для перекодировки по таблице XLATAB. В регистре AL устанавливается в результате значение шест.34. Затем процедура выделяет вторую шест. цифру F и перекодирует ее в шест.46. В результате обработки получается шест.3446, что отображается на экране как 4F.

Так как функция DOS для вывода на экран (шест.40) рассматривает шест.1A как конец файла, то в программе это значение заменяется на пробел. Программа, использующая для вывода на экран функцию DOS (шест.09), должна заменять символ ограничитель '$' на пробел.

Существует много различных способов преобразования шест. цифр в ASCII-символы. Можно поэкспериментировать с операциями сдвига и сравнения.

ПРОГРАММА: СОРТИРОВКА ЭЛЕМЕНТОВ ТАБЛИЦЫ

Часто возникает необходимость сортировки элементов таблицы в восходящем или нисходящем порядке. Например, пользователю может потребоваться список наименований товара в алфавитном порядке или список общих цен в нисходящей последовательности. Обычно, табличные данные не определяются как в предыдущей программе, а загружаются с клавиатуры или с диска. Данный раздел посвящен сортировке элементов таблицы, что касается различных применений, включающих сортировку записей на дисках, то здесь возможны более сложные программы.

Существует несколько алгоритмов сортировки таблиц от неэффективных, но понятных, до эффективных и непонятных. Программа сортировки, предлагаемая в данном разделе, весьма эффективна и может применяться для большинства табличных сортировок. Конечно, если не проверить различные алгоритмы сортировок, то даже самая неэффективная программа может показаться работающей со скоростью света. Но цель данной книги - показать технику ассемблера, а не сортировки. Основной подход заключается в сравнении соседних элементов таблицы. Если первый элемент больше второго, то элементы меняются местами. Таким образом выполняется сравнение элементов 1 со 2, 2 с 3 и т.д. до конца таблицы с перестановкой элементов там, где это необходимо. Если в проходе были сделаны перестановки, то весь процесс повторяется с начала таблицы т.е. сравниваются снова элементы 1-2, 2-3 и т.д. Если в проходе не было перестановок, то таблица отсортирована и можно прекратить процесс.

Ниже приведен алгоритм, в котором переменная SWAP является индикатором: была перестановка элементов (YES) или нет (NO):

```
G10:   Определить адрес последнего элемента
G20:   Установить SWAP=NO
       Определить адрес первого элемента
G30:   Элемент > следующего элемента?
           Да:  Представить элементы
                Установить SWAP=YES
       Перейти к следующему элементу
       Конец таблицы?
```

Нет: Перейти на G30
Да: SWAP=YES?
Да: Перейти на G20 (повторить сорт.)
Нет: Конец сортировки

Программа, показанная на рис.14.6, обеспечивает ввод с клавиатуры до 30 имен, сортировку введенных имен в алфавитном порядке и вывод на экран отсортированного списка имен.

```
      page    60,132
TITLE   NMSORT  (EXE) Ввод и сортировка имен
; --------------------------------------------
STACK   SGMENT  PARA STACK 'Stack'
      DW      32 DUP(?)
STACK   ENDS
; --------------------------------------------
DATASG  SEGMENT PARA 'Data'
NAMEPAR LABEL   BYTE           ;Имя списка параметров:
MAXNLEN DB      21             ; макс. длина
NAMELEN DB      ?              ; число введенных символов
NAMEFLD DB      21 DUP(' ')    ; имя

CRLF    DB      13, 10, '$'
ENDADDR DW      ?
MESSG1  DB      'Name?', '$'
NAMECTR DB      00
NAMETAB DB      30 DUP(20 DUP(' ')) ;Таблица имен
NAMESAV DB      20 DUP(?), 13, 10, '$'
SWAPPED DB      00
DATA    ENDS
; --------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
      ASSUME CS:CODESG,DS:DATDSG,SS:STACK,ES:DATASG
      PUSH    DS
      SUB     AX,AX
      PUSH    AX
      MOV     AX,DATASG
      MOV     DS,AX
      MOV     ES,AX
      CLD
      LEA     DI,NAMETAB
      CALL    Q10CLR          ;Очистить экран
      CALL    Q20CURS         ;Установить курсор
A20LOOP:
      CALL    B10READ         ;Ввести имя с клавиатуры
      CMP     NAMELEN,00      ;Есть ли еще имена?
      JZ      A30             ; нет - идти на сортировку
      CMP     NAMECTR,30      ;Введено 30 имен?
      JE      A30             ; да - идти на сортировку
      CALL    D10STOR         ;Записать имя в таблицу
      JMP     A20LOOP
A30:                          ;Конец ввода имен
      CALL    Q10CLR          ;Очистить экран
      CALL    Q20CURS         ; и установить курсор
      CMP     NAMECTR,01      ;Введено менее 2 имен?
      JBE     A40             ; да - выйти
      CALL    G10SORT         ;Сортировать имена
      CALL    K10DISP         ;Вывести результат на экран
```

```
A40:    RET                    ;Завершить программу
BEGIN   ENDP
;              Ввод имен с клавиатуры?
;              ---------------------
B10READ PROC
        MOV     AH,09
        LEA     DX,MESSG1       ;Вывести текст запроса
        INT     21H
        MOV     AH,0AH
        LEA     DX,NAMEPAR      ;Ввести имя
        INT     21H
        MOV     AH,09
        LEA     DX,CRLF         ;Вывести CRLF
        INT     21H

        MOV     BH,00           ;Очистить поле после имени
        MOV     BL,NAMELEN      ;Получить счетчик символов
        MOV     CX,21
        SUB     CX,BX           ;Вычислить оставшуюся длину
B20:
        MOV     NAMEFLD[BX],20H ;Установить символ пробела
        INC     BX
        LOOP    B20
        RET
B10READ ENDP
;              Запись имени в таблицу:
;              ---------------------
D10STOR PROC
        INC     NAMECTR         ;Число имен в таблице
        CLD
        LES     SI,NAMEFLD
        MOV     CX,10
        REP MOVSV               ;Переслать имя в таблицу
        RET
D10STOR ENDP
;              Сортировка имен в таблице:
;              ------------------------
G10SORT PROC
        SUB     DI,40           ;Установить адреса останова
        MOV     ENDADDR,DI
G20:
        MOV     SWAPPED,00      ;Установить начало
        LEA     SI,NAMETAB      ; таблицы
G30:
        MOV     CX,20           ;Длина сравнения
        MOV     DI,SI
        ADD     DI,20           ;Следующее имя для сравнения
        MOV     AX,DI
        MOV     BX,SI
        REPE CMPSB              ;Сравнить имя со следующим
        JBE     G40             ; нет перестановки
        CALL    H10XCHG         ; перестановка
G40:
        MOV     SI,AX
        CMP     SI,ENDADDR      ;Конец таблицы?
        JBE     G30             ; нет - продолжить
        CMP     SWAPPED,00      ;Есть перестановки?
        JNZ     G20             ; да - продолжить,
        RET                     ; нет - конец сортировки
G10SORT ENDP
;              Перестановка элементов таблицы:
```

```
;           ------------------------------
H10XCHG PROC
      MOV    CX,10
      LEA    DI,NAMESAV
      MOV    SI,BX
      REP MOVSW            ;Сохранить меньший элемент

      MOV    CX,10
      MOV    DI,BX
      REP MOVSW            ;Переслать больший элемент
                           ; на место меньшего
      MOV    CX,10
      LEA    SI,NAMESAV
      REP MOVSW            ;Переслать сохраненный
                           ; элемент на место большего
      MOV    SWAPPED,01    ;Признак перестановки
      RET
H10XCHG ENDP
;           Вывод на экран отсортированные имена:
;           -----------------------------------
K10DISP PROC
      LEA    SI,NAMETAB
K20:
      LEA    DI,NAMESAV     ;Начальный адрес таблицы
      MOV    CX,10
      REP MOVSV
      MOV    AH,09
      LEA    DX,NAMESAV
      INT    21H           ;Вывести на экран
      DEC    NAMECTR        ;Это последний элемент?
      JNZ    K20           ; нет - повторить цикл,
      RET                  ; да - выйти
K10DISP ENDP
;           Очистка экрана:
;           --------------
Q10CLR  PROC
      MOV    AX,0600H
      MOV    BH,61H        ;Цвет (07 для ч/б)
      SUB    CX,CX
      MOV    DX,184FH
      INT    10H
      RET
Q10CLR  ENDP
;           Установка курсора:
;           -----------------
Q20CURS PROC
      MOV    AH,02
      SUB    BH,BH
      SUB    DX,DX         ;Установить курсор в 00,00
      INT    10H
      RET
Q20CURS ENDP

CODESG  ENDS
      END    BEGIN
```

Рис.14.6. Сортировка таблицы имен

ОПЕРАТОРЫ ТИПА, ДЛИНА И РАЗМЕРА

Ассемблер содержит ряд специальных операторов, которые могут оказаться полезными при программировании. Например, при изменении длины таблицы придется модифицировать программу (для нового определения таблицы) и процедуры, проверяющие конец таблицы. В этом случае использование операторов TYPE (тип), LENGTH (длина) и SIZE (размер) позволяют уменьшить число модифицируемых команд.

Рассмотрим определение следующей таблицы из десяти слов:

```
TABLEX    DW   10 DUP(?) ;Таблица из 10 слов
```

Программа может использовать оператор TYPE для определения типа (DW в данном случае), оператор LENGTH для определения DUP-фактора (10) и оператор SIZE для определения числа байтов (10 x 2 = 20). Следующие команды иллюстрируют три таких применения:

```
MOV  AX,TYPE TABLEX      ;AX=0002
MOV  BX,LENGTH TABLEX    ;BX=000A  (10)
MOV  CX,SIZE TABLEX      ;CX=0014  (20)
```

Значения LENGTH и SIZE можно использовать для окончания табличного поиска или сортировки. Например, если регистр SI содержит продвинутый адрес таблицы при осуществлении поиска, то проверка на конец таблицы может быть следующий:

```
CMP  SI,SIZE TABLEX
```

В главе 23 "Справочник по директивам ассемблера" дается детальное описание операторов TYPE, LENGTH и SIZE.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Для большинства применений, определяйте таблицы, имеющие родственные элементы одной длины и формата данных.

- Стройте таблицы на основе форматов данных. Например, элементы могут быть символьные или числовые длиной один, два и более байтов каждый. Может оказаться более практичным определение двух таблиц: одна, например, для трехсимвольных значений номеров, а другая для двухбайтовых значений цен единиц товара. В процессе поиска адрес элементов таблицы номеров должен увеличиваться на 3, а адрес элементов таблицы цен - на 2. Если сохранить число выполненных циклов при поиске на равно, то, умножив это число на 2 (SHL сдвиг влево на один бит), получим относительный адрес искомого значения цены. (Начальное значение счетчика циклов должно быть равно -1).

- Помните, что DB позволяет определять значения, не превышающие 256, а DW записывает байты в обратной последовательности. Команды CMP и CMPSW предполагают, что байты в сравниваемых словах имеют обратную последовательность.

- Если таблица подвергается частым изменениям, или должна быть доступна нескольким программам, то запишите ее на диск. Для внесения изменений в таблицу можно разработать специальную программу модификации. Любые программы могут загружать таблицу с диска и при обновлениях таблицы сами программы не нуждаются в изменениях.

- Будьте особенно внимательны при кодировке сортирующих программ. Пользуйтесь трассировкой для тестирования, так как малейшая ошибка может

привести к непредсказуемым результатам.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

14.1. Определите таблицу, которая содержит имена дней недели, начиная с воскресения.

14.2 Предполагая, что воскресенье равно 1, напишите команды прямого доступа к таблице, определенной в вопросе 14.1. используйте любые подходящие имена.

14.3 Определите три отдельных связанных таблицы, содержащих следующие данные:

а) числовые элементы: 06, 10, 14, 21, 24;

б) элементы наименований: видеокассеты, приемники, модемы, клавиатуры, дискеты;

в) цены: 93.95, 82.25, 90.67, 85.80, 13.85.

14.4 Составьте программу, позволяющую вводить числовой элемент (ITEMIN) и количество (QTYIN) с клавиатуры. Используя таблицу из вопроса 14.3, разработайте программу табличного поиска элемента равного ITEMIN. Выделите из таблиц наименование и цену. Рассчитайте величину стоимости (Количество x Цена) и выдайте на экран наименование и стоимость.

14.5 Используя описание таблицы из вопроса 14.3, составьте процедуры: а) пересылающую содержимое одной таблицы в новую (пустую) таблицу; б) сортирующую содержимое новой таблицы в восходящей последовательности.

ГЛАВА 15 Дисковая память I: Организация

Ц е л ь: Рассмотреть основные форматы записей в памяти на твердом диске (винчестере) и на дискете, включая оглавление и таблицу распределения файлов.

ВВЕДЕНИЕ

Диск является распространенным средством для более или менее долговременного хранения данных. Процессы обработки данных на твердом диске (винчестере) аналогичны процессам для гибких дисков (дискет), за исключением того, что возможно потребуется обеспечить пути для доступа к многочисленным подоглавлениям винчестера. Для обработки файлов полезно ознакомиться с организацией дисковой памяти. Каждая сторона стандартной 5 1/4 дюймовой дискеты содержит 40 концентрических дорожек, пронумерованных от 00 до 39. На каждой дорожке форматируется восемь или девять секторов по 512 байтов каждый.

Данные записываются на диск в виде файлов, аналогично тому, как вы записываете ассемблерные программы. Хотя на типы данных, которые можно хранить в файле, не существует каких-либо ограничений, типичный пользовательский файл содержит списки заказчиков, описи товаров и предложений или списки имен и адресов. Каждая запись содержит информацию о конкретном заказчике или описание товара. Внутри файла все записи имеют одинаковую длину и формат. Запись может содержать одно или несколько полей. Файл заказчиков, например, может состоять из записей, в которые входит номер заказчика, имя заказчика и долговой баланс. Эти записи могут

быть расположены в порядке возрастания номеров заказчиков следующим образом:

```
---T---T-----¬---T---T-----¬---T---T-----¬   ---T---T-----¬
¦#1¦имя¦сумма¦¦#2¦имя¦сумма¦¦#3¦имя¦сумма¦...¦#n¦имя¦сумма¦
L--+---+------L--+---+------L--+---+------   L--+---+------
```

Для программирования дисковых файлов следует в общих чертах ознакомится только с концепцией и терминологией. Если в данной главе размеры диска не указываются, то предполагается диск 5 1/4" формата.

ЕМКОСТЬ ДИСКА

Емкость гибких дисков:

Версия DOS и тип диска	Число дорожек на стороне	Число секторов на дорожке	Число байтов в секторе	Всего на двух сторонах
До DOS 2.0	40	8	512	327 680
DOS 2.0 и после	40	9	512	368 640
Высокая плотность	80	15	512	1 228 800
3 1/2"	80	9	512	737 280

Емкость твердых дисков:

Тип диска	Число дорожек на стороне	Число секторов на дорожке	Число байтов в секторе	Всего на 4-х сторонах
10 мегабайт	306	17	512	10 653 696
20 мегабайт	614	17	512	21.377.024

Указание стороны (головки), дорожки или сектора на диске осуществляется по номеру. Для стороны и дорожки отсчет ведется с 0, а для сектора - с 1.

ОГЛАВЛЕНИЕ ДИСКА (КАТАЛОГ)

Для того, чтобы организовать хранение информации на диске, операционная система DOS резервируют определенные сектора для своих нужд. Организация данных на дискете или на твердом диске существенно зависит от их емкости. Форматированная двухсторонняя дискета с девятью секторами на дорожке содержит следующую системную информацию:

Сторона	Дорожка	Сектор	
0	0	1	Запись начальной загрузки
0	0	2-3	Таблица распределения файлов (FAT)
0	0	4-7	Каталог
1	0	1-3	Каталог
1	0	4 ...	Файлы данных

Область записей данных начинается с третьего сектора на 1-й стороне 0-й дорожки и продолжается до девятого сектора. Следующие записи заносятся на 0-ю сторону 1-й дорожки, затем на 1-ю сторону 1-й дорожки, затем на 0-ю сторону 2-й дорожки и т.д. Такая особенность заполнения дисковой памяти на

противоположных дорожках снижает число перемещений головки дисковода. Данный метод используется как для гибких, так и для твердых дисков.

При использовании утилиты FORMAT /S для форматизации дискеты, модули DOS IBMBIO.COM и IBMDOS.COM записывается в первые сектора области данных.

Все файлы, даже меньшие 512 байт (или кратные 512), начинаются на границе сектора. Для каждого файла DOS создает на нулевой дорожке диска элемент оглавления. Каждый такой элемент описывает имя, дату, размер и расположение файла на диске. Элементы оглавления имеют следующий формат:

Байт Назначение

0-7 Имя файла, определяемое из программы, создавшей данный файл. Первый байт может указывать на статус файла: шест.00 обозначает, что данный файл не используется, шест.E5 - файл удален, шест.2E - элемент подоглавления.

8-10 Тип файла

11 Атрибут файла, определяющий его тип:
шест.00 - обычный файл;
шест.01 - файл можно только читать;
шест.02 - "спрятанный" файл;
шест.04 - системный файл DOS;
шест.08 - метка тома;
шест.10 - подоглавление;
шест.20 - архивный файл (для твердого диска).

12-21 Зарезервировано для DOS.

22-23 Время дня, когда файл был создан или последний раз изменялся, в следующим двоичном формате:

|чччччммммммссссс|

24-25 Дата создания или последнего изменения файла, сжатая в два слова в следующем двоичном формате:

|ггггггг|ммммддддд|

где год начинается с 1980 и может принимать значения от 0 до 119, месяц - от 1 до 12, а день - от 1 до 31.

26-27 Начальный кластер файла. Относительный номер последних двух секторов каталога. Первый файл данных (без COM-модулей DOS) начинается на относительном кластере 002. Текущая сторона, дорожка и кластер зависят от емкости диска.

28-31 Размер файла в байтах. При создании файла DOS вычисляет и записывает размер файла в это поле.

Все поля в каталоге диска, превышающие один байт, записываются в обратной последовательности байтов.

ТАБЛИЦА РАСПРЕДЕЛЕНИЯ ФАЙЛОВ

Назначение таблицы распределения файлов (FAT - File Allocation Table) - распределение дискового пространства для файлов. Если вы создаете новый файл или изменяете существующий, то DOS меняет элементы таблицы файлов в соответствии с расположением файла на диске. Запись начальной загрузки находится на секторе 1, далее на секторе 2 начинается FAT. FAT содержит элементы для каждого кластера, длина элементов FAT зависит от устройства дисковой памяти. Кластер для односторонних дискет представляет собой один сектор, для двухсторонних дискет - смежную пару секторов. Одно и то же число элементов в FAT определяет в два раза больше данных для двухсторонних дискет, чем для односторонних.

Первые байты FAT определяют тип устройства:

FE Односторонняя на 8 секторов
FC Односторонняя на 9 секторов
FF Двухсторонняя на 8 секторов
FD Двухсторонняя на 9 секторов
F9 Повышенная емкость (1,2 мегабайта)

F8 Твердый диск

Второй и третий байты пока содержат FFFF. В следующей таблице показана организация данных для нескольких типов устройств (приведены начальные и конечные номера секторов). Колонка "Кластер" представляет число секторов в кластере:

Устройство диска	Запись нач.загр.	FAT	Каталог	Кластер
Односторонний, 8 секторов	1	2-3	4-7	1
Односторонний, 9 секторов	1	2-5	6-9	1
Двухсторонний, 8 секторов	1	2-3	4-10	2
Двухсторонний, 9 секторов	1	2-5	6-12	2
Повышенная емкость (1,2 М)	1	2-15	16-29	1
Твердый диск XT	1	2-17	18-49	8
Твердый диск AT	1	2-838	4-115	4

Начиная с четвертого байта, элементы FAT определяют сектора. Каждый такой элемент имеет длину 12 битов. (В версии DOS 3 и старше элементы FAT для твердого диска могут иметь длину 16 битов). Два первых элемента FAT, известные как относительные сектора 000 и 001, соответственно, указывают на два последних сектора оглавления, определяя его размер и формат. Первый файл данных начинается на относительном секторе 002. Каждый элемент FAT состоит из трех шест.цифр (12 битов), которые указывают на характер использования конкретного сектора:

000 свободный кластер,
nnn относительный номер следующего кластера для файла,
FF7 неиспользуемый кластер (сбойная дорожка),
FFF последний кластер файла.

Предположим, например, что дискета содержит только один файл с именем PAYROLL.ASM, занимающий относительные сектора 002, 003 и 004. Элемент оглавления для этого файла содержит имя файла PAYROLL, тип - ASM, шест.00 для обычного файла, дату создания, 002 - номер первого относительного сектора файла и размер файла в битах. Таблица FAT в этом случае может выглядеть следующим образом (кроме того, что в каждой паре байты в обратной последовательности):

```
Элемент FAT:        |FDF|FFF|003|004|FFF|000|000|...|000|
Относительн.сектор:   0   1   2   3   4   5   6  ...конец
```

Первые два элемента FAT указывают расположение каталога на относительных секторах 000 и 001. Для ввода рассматриваемого файла в память, система выполняет следующие действия:

1. DOS получает доступ к дискете и ищет в каталоге имя PAYROLL и тип ASM.
2. Затем DOS определяет по каталогу положение первого относительного сектора файла (002) и загружает содержимое этого сектора в буферную область в основной памяти.
3. Номер второго сектора DOS получает из элемента FAT, соответствующего относительному сектору 002. Из диаграммы,

приведенной выше, видно, что этот элемент содержит 003. Это обозначает, что файл продолжается в относительном секторе 003. DOS загружает содержимое этого сектора в буфер в основной памяти.

4. Номер третьего сектора DOS получает из элемента FAT, соответствующего относительному сектору 003. Этот элемент содержит 004, значит файл продолжается в относительном секторе 004. DOS загружает содержимое этого сектора в буфер в основной памяти.

5. Элемент FAT для относительного сектора 004 содержит шест.FFF, что свидетельствует о том, что больше нет данных для этого файла.

Элемент каталога содержит номер начального кластера для каждого файла, а FAT - шест.трехзначные элементы, указывающие на расположение каждого дополнительного кластера, если он имеется. Для того, чтобы указать, например, что файл содержит все записи только в первом кластере, таблица FAT должна содержать шест.FFF в элементе, представляющем первый относительный кластер.

В качестве простого примера рассмотрим элемент каталога, указывающий, что некоторый файл начинается в относительном кластере 15. Для локализации первого элемента таблицы FAT необходимо:

- Умножить 15 на 1,5, получим 22,5.
- Выполнить выборку содержимого байтов 22 и 23 из FAT. Предположим, что они содержат F*FF.
- Переставить байты: FFF*.
- Так как номер 15-нечетный, то первые три цифры - FFF указывают на отсутствие других кластеров для данного файла.

Теперь рассмотрим файл, который занимает четыре кластера, начинающихся с номера 15. Таблица FAT, начиная с байта 22 и далее, в этот раз показана в правильной обратной последовательности байтов в парах:

6* 01 17 80 01 FF*F

Для того, чтобы найти первый элемент FAT, необходимо умножить 15 на 1,5, получим 22,5, и выбрать содержимое байтов 22 и 23, как в предыдущем примере. В этот раз эти байты содержат 6*01, что после перестановки байт даст 016*. Так как 15-число нечетное, то используются первые три цифры 016. Второй кластер для файла, следовательно, имеет номер 016.

Для того, чтобы найти третий кластер, необходимо умножить 16 на 1,5 получим 24. Затем следует выбрать содержимое байтов 24 и 25 таблицы FAT. Значение 1780 после перестановки байтов даст 8017. Так как число 16 четное, то используются последние три цифры 017. Третий кластер для файла имеет номер 017.

Для того, чтобы найти четвертый кластер, необходимо умножить 17 на 1,5, получим 25.5. Затем следует выбрать содержимое байтов 25 и 26 таблицы FAT. Значение 8001 после перестановки байтов даст 0180. Так как число 17 нечетное, то используются первые три цифры 018. Четвертый кластер для файла имеет номер 018.

При использовании этой же процедуры для локализации содержимого следующего элемента FAT по относительным адресам 27 и 28, получим FF*F, что после перестановки даст *FFF. Так как число 18 четное, используются последние три цифры FFF, что обозначает последний элемент.

Как было ранее сказано, все файлы начинаются на границе кластера. Кроме того, совсем не обязательно файл должен храниться в соседних кластерах, он может быть разбросан на диске по разным секторам.

Если в программе необходимо определить тип установленного диска, то можно обратиться к таблице FAT непосредственно, или, что предпочтительней, использовать функцию DOS 1BH или 1CH.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Независимо от размеров все файлы начинаются на границе кластера.

- Оглавление (каталог) содержит для каждого файл на диске элементы, определяющие имя, тип, атрибуты, дату, начальный сектор и размер файла.

- Таблица распределения файлов (FAT) содержит один элемент для каждого кластера в каждом файле.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

15.1. Какую длину в байтах имеет стандартный сектор?

15.2. Где расположена запись начальной загрузки?

15.3. Как обозначаются в оглавлении удаленные файлы?

15.4. Какие дополнительные действия выполняются при форматизации дискеты по команде DOS FORMAT /S?

15.5. Где и каким образом обозначается в таблице FAT, что устройством является твердый диск?

15.6. Имеется файл размером 2890 (десятичное) байтов: а) Где хранит система размер файла? б) Как выражается этот размер в шестнадцатиричном формате? в) Покажите значение в том виде, как оно записывается системой.

ГЛАВА 16 Дисковая память II: Функции базовой версии DOS

Ц е л ь: Раскрыть основные требования к программированию функций базовой версии DOS для обработки дисковых файлов.

ВВЕДЕНИЕ

В начале данной главы рассматриваются функции базовой версии DOS, определяющие блок управления файлом (FCB), а затем будут показаны возможности создания и обработки дисковых файлов последовательным и прямым доступом. Все рассматриваемые операции были введены в первых версиях DOS и возможны во всех последующих версиях.

Обработка дисковых файлов в базовой DOS включает определение блока управления файлом (FCB - file control block), который описывает файл и его записи. Передача адреса блока FCB в DOS обязательна для всех дисковых операций ввода-вывода. Новых команд ассемблера в данной главе не потребуется.

Управление вводом и выводом осуществляется специальными прерываниями. Запись файла на диск требует, чтобы прежде он был "создан" и DOS смогла сгенерировать соответствующий элемент в оглавлении. Когда все записи файла будут записаны, программа должна "закрыть" файл, так, чтобы DOS завершила обработку оглавления. Чтение файла требует, чтобы он был сначала "открыт" для того, чтобы убедиться в его существовании. Так как записи имеют фиксированную длину и в силу соответствующей организации оглавления, обработка записей дискового файла может осуществляться как последовательно, так и произвольно.

Метод доступа к дисковой памяти, поддерживающий использование оглавления, "блокирование" и "разблокирование" записей, обеспечивается прерыванием DOS 21H. Более низкий уровень, обеспечивающий абсолютную адресацию дисковых секторов, также через DOS, выполняется посредством прерываний 25H и 26H. Самый низкий уровень обеспечивается прерыванием BIOS 13H, которое позволяет выполнить произвольную адресацию в дисковой памяти по номеру дорожки и сектора. Методы DOS осуществляют некоторую предварительную обработку до передачи управления в BIOS. В гл.17 объясняется применение предпочтительных функций расширенного DOS 2, а гл.18 представляет основные дисковые операции в BIOS. Напоминание: Термин кластер определяет один или более секторов с данными в зависимости от дискового устройства.

БЛОК УПРАВЛЕНИЯ ФАЙЛОМ (FCB)

Для выполнения операций ввода-вывода на диске в базовой DOS необходимо в области данных определить блок FCB. Блок FCB не поддерживает путь доступа к файлу, поэтому он используется главным образом для обработки файлов в текущей директории. Блок FCB содержит описание файла и его записей в приведенном ниже формате. Пользователь должен инициализировать байты 0-15 и 32-36, байты 16-31 устанавливаются DOS:

Байты	Назначение
0	Указывает дисковод: 01 для дисковода A, 02 для B и т.д.
1-8	Имя файла, выравненное по левой границе с конечными пробелами, если имя меньше 8 байт. Поле может содержать зарезервированные имена, например, LPT1 для принтера.
9-11	Тип файла для дополнительной идентификации, например, DTA или ASM. Если тип файла меньше трех байт, то он должен быть выравнен по левой границе и дополнен конечными пробелами. DOS хранит имя и тип файла в оглавлении.
12-13	Номер текущего блока. Блок содержит 128 записей. Для локализации конкретной записи используется номер текущего блока и номер текущей записи (байт 32). Первый блок файла имеет номер 0, второй - 1 и т.д. Операция открытия файла устанавливает в данном поле 0.
14-15	Логический размер записи. Операция открытия инициализирует размер записи значением 128 (шест.80). После открытия и перед любой операцией чтения или записи можно устанавливать в данном поле любое требуемое значение длины записи.
16-19	Размер файла. При создании файла DOS вычисляет и записывает это значение (произведение числа записей на размер записей) в оглавление. Операция открытия выбирает размер файла из оглавления и заносит его в данное поле. Программа может читать это поле, но не может менять его.
20-21	Дата. При создании или последней модификации файла DOS записывает дату в оглавление. Операция открытия выбирает дату из оглавления и заносит в данное поле.
22-31	Зарезервировано для DOS.
32	Текущий номер записи. Данное поле содержит текущий номер записи (0-127) в текущем блоке (см. байты 12-13). Система использует текущие значения блока и записи для локализации записи в дисковом файле. Обычно номер начальной записи в данном поле - 0, но его можно заменить для начала последовательной обработки на любое значение от 0 до 127.
33-36	Относительный номер записи. Для произвольного доступа при операциях чтения или записи данное поле должно содержать относительный номер записи. Например, для произвольного чтения

записи номер 25 (шест.19), необходимо установить в данном поле шест.19000000. Произвольный доступ характеризуется тем, что система автоматически преобразует относительный номер записи в текущие значения блока и записи. Ввиду ограничения на максимальный размер файла (1.073.741.824 байтов), файл с короткими записями может содержать больше записей и иметь больший относительный номер записи. Если размер записи больше 64, то байт 36 всегда содержит 00.

Помните, что числовые значения в словах и двойных словах записываются в обратной последовательности байтов.

Блоку FCB предшествует необязательное семибайтовое расширение, которое можно использовать для обработки файлов со специальными атрибутами. Для использования расширения необходимо закодировать в первом байте шест.FF, во втором - атрибут файла, а в остальных пяти байтах шест. нули.

ИСПОЛЬЗОВАНИЕ БЛОКА FCB ДЛЯ СОЗДАНИЯ ФАЙЛА НА ДИСКЕ

Для ссылки на каждый дисковый файл программа должна содержать правильно составленный блок управления файлом. Операции ввода-вывода на диск требуют установки адреса блока FCB в регистре DX. Доступ к полям блока FCB осуществляются по этому адресу с помощью регистровой пары DS:DX. Для создания нового файла программа использует функцию шест.16 в прерывании DOS INT 21H следующим образом:

```
MOV  AH,16H     ;Создание
LEA  DX,FCBname ; дискового файла
INT  21H        ;Вызов DOS
```

DOS осуществляет поиск имени файла и тип файла, взятого из соответствующих полей FCB, в оглавлении. Если элемент оглавления, содержащий необходимое имя (и тип), будет найдено, то DOS очищает найденный элемент для нового использования, если такой элемент не будет найден, то DOS ищет свободный элемент. Затем операция устанавливает размер файла в 0 и "открывает" файл. На этапе открытия происходит проверка доступного дискового пространства, результат такой проверки устанавливается в регистре AL:

```
00   На диске есть свободное пространство
FF   На диске нет свободного пространства.
```

При открытии также устанавливается в блок FCB номер текущего блока - 0 и размер записей (по умолчанию) - 128 (шест.80) байтов. Прежде, чем начать запись файла, можно заменить это значение по умолчанию на требуемый размер записей.

Для определения выводной записи необходимо прежде обеспечить начальный адрес этой записи в область передачи данных (DTA - disk trausfer area). Так как блок FCB содержит размер записей, то в DTA не требуется устанавливать ограничитель конца записи. Затем с помощью функции шест.1A необходимо сообщить DOS адрес DTA. В любой момент времени может быть активен только один DTA. В следующем примере инициализируется адрес DTA:

```
MOV  AH,1AH      ;Установка адреса
LEA  DX,DTAname  ; DTA
INT  21H         ;Вызов DOS
```

Если программа обрабатывает только один дисковой файл, то должна быть только одна установка адреса DTA для всего выполнения. При обработке

нескольких файлов программа должна устанавливать соответствующий адрес DTA непосредственно перед каждой операцией чтения или записи.

Для последовательной записи на диск существует функция шест.15:

```
MOV  AH,15          ;Последовательная
LEA  DX,FCBname     ; запись
INT  21H            ;Вызов DOS
```

Операция записи использует информацию из блока FCB и адрес текущего буфера DTA. Если длина записи равна размеру сектора, то запись заносится на диск. В противном случае записи заполняют буфер по длине сектора и затем буфер записывается на диск. Например, если длина каждой записи составляет 128 байтов, то буфер заполняется четырьмя записями (4*128=512) и затем буфер записывается в дисковой сектор.

После успешного занесения записи на диск DOS увеличивает в блоке FCB размер файла на размер записи и текущий номер записи на 1. Когда номер текущей записи достигает 128, происходит сброс этого значения в 0 и в FCB увеличивается номер текущего блока на 1. Операция возвращает в регистре AL следующие коды:

- 00 Успешная запись.
- 01 Диск полный.
- 02 В области DTA нет места для одной записи.

Когда запись файла завершена, можно, хотя и не всегда обязательно, записать маркер конца файла (шест.1A). Для закрытия файла используется функция шест.10:

```
MOV  AH,10H         ;Закрыть
LEA  DX,FCBname     ; файл
INT  21H            ;Вызов DOS
```

Эта операция записывает на диск данные, которые еще остались в дисковом буфере DOS и изменяет в соответствующем элементе оглавления, дату и размер файла. В регистре AL возвращаются следующие значения:

- 00 Успешная запись.
- FF Описание файла оказалось в неправильном элементе оглавления (возможно в результате смены дискеты).

ПРОГРАММА: ИСПОЛЬЗОВАНИЕ FCB ДЛЯ СОЗДАНИЯ ФАЙЛА НА ДИСКЕ

```
        page    60,132
TITLE   FCBCREAT (EXE) Использование FCB для создания файла
;-------------------------------------------------------
STACKSG SEGMENT PARA STACK 'Stack'
        DW      80 DUP(?)
STACKSG ENDS
;-------------------------------------------------------
DATASG  SEGMENT PARA 'Data'
RECLEN  EQU     32
NAMEPAR LABEL   BYTE            ;Список параметров:
MAXLEN  DB      RECLEN          ; макс.длина имени
NAMELEN DB      ?               ; число введенных символов
NAMEDTA DB      RECLEN DUP(' ') ; область передачи (DTA)

FCBREC  LABEL   BYTE            ;FCB для дискового файла
```

```
FCBDRIV DB      04              ; дисковод D
FCBNAME DB      'NAMEFILE'      ; имя файла
FCBEXT  DB      'DAT'           ; тип файла
FCBBLK  DW      0000            ; номер текущего блока
FCBRCSZ DW      ?               ; размер логической записи
FCBFLSZ DD      ?               ; размер файла (DOS)
        DW      ?               ; дата (DOS)
        DT      ?               ; зарезервировано (DOS)
FCBSQRC DB      00              ; номер текущей записи
        DD      ?               ; относительный номер

CRLF    DB      13,10,'$'
ERRCDE  DB      00
PROMPT  DB      'Name? ','$'
ROW     DB      01
OPNMSG  DB      '*** Open error ***',  '$'
WRTMSG  DB      '*** Write error ***', '$'
DATASG  ENDS
; --------------------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
        ASSUME CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASG
        PUSH    DS
        SUB     AX,AX
        PUSH    AX
        MOV     AX,DATASG
        MOV     DS,AX
        MOV     ES,AX
        MOV     AX,0600H
        CALL    Q10SCR          ;Очистить экран
        CALL    Q20CURS         ;Установить курсор
        CALL    C10OPEN         ;Открыть, установить DTA
        CMP     ERRCDE,00       ;Есть место на диске?
        JZ      A20LOOP         ; да - продолжить,
        RET                     ; нет - вернуться в DOS
A20LOOP:
        CALL    D10PROC
        CMP     NAMELEN,00      ;Конец ввода?
        JNE     A20LOOP         ; нет - продолжить,
        CALL    G10CLSE         ; да - закрыть файл
        RET                     ; и вернуться в DOS
BEGIN   ENDP
;               Открытие дискового файла:
;               ------------------------
C10OPEN PROC    NEAR
        MOV     AH,16H          ;Функция создания файла
        LEA     DX,FCBREC
        INT     21H
        CMP     AL,00           ;Есть место на диске?
        JNZ     C20             ; нет - ошибка

        MOV     FCBRCSZ,RECLEN  ;Размер записи (EQU)
        LEA     DX,NAMEDTA      ;Загрузить адрес DTA
        MOV     AH,1AH
        INT     21AH
        RET
C20:
        LEA     DX,OPNMSG       ;Сообщение об ошибке
        CALL    X10ERR
        RET
C10OPEN ENDP
```

```
;              Ввод с клавиатуры:
;              -----------------
D10PROC PROC    NEAR
       MOV     AH,09          ;Функция вывода на экран
       LEA     DX,PROMPT       ;Выдать запрос
       INT     21H

       MOV     AH,0AH         ;Функция ввода
       LEA     DX,NAMEPAR      ;Ввести имя файла
       INT     21H
       CALL    E10DISP        ;Прокрутка на экране

       CMP     NAMELEN,00      ;Имя введено?
       JNE     D20            ; да - продолжить,
       RET                    ; нет - выйти
D20:
       MOV     BH,00          ;Заменить символ Return
       MOV     BL,NAMELEN
       MOV     NAMEDTA[BX],' ' ;Записать пробел
       CALL    F10WRIT        ;Вызвать
                              ; подпрограмму записи
       CLD
       LEA     DI,NAMEDTA      ;Очистить
       MOV     CX,RECLEN / 2   ; поле
       MOV     AX,2020H        ; имени
       REP STOSW
       RET                    ;Выйти
D10PROC ENDP
;              Прокрутка и установка курсора:
;              -----------------------------
E10DISP PROC    NEAR
       MOV     AH,09          ;Функция вывода на экран
       LEA     DX,CRLF        ;CR/LF
       INT     21H            ;Вызов DOS
       CMP     ROW,18          ;Последняя строка экрана?
       JAE     E20            ; да - обойти,
       INC     ROW             ; нет - увеличить строку
       RET
E20:
       MOV     AX,0601H        ;Прокрутка на 1 строку
       CALL    Q10SCR
       CALL    Q20CURS         ;Установить курсор
       RET
E10DISP ENDP
;              Запись на диск:
;              --------------
F10WRIT PROC    NEAR
       MOV     AH,15H         ;Функция записи
       LEA     DX,FCBREC
       INT     21H
       CMP     AL,00          ;Запись без ошибок?
       JZ      F20            ; да
       LEA     DX,WRTMSG       ; нет -
       CALL    X10ERR         ; выдать сообщение
       MOV     NAMELEN,00
F20:   RET
F10WRIT ENDP
;              Закрытие дискового файла:
;              ------------------------
G10CLSE PROC    NEAR
       MOV     NAMEDTA,1AH     ;Установить EOF
```

```
       CALL    F10WRIT
       MOV     AH,10H          ;Функция закрытия
       LEA     DX,FCBREC
       INT     21H
       RET
G10CLSE ENDP
;              Прокрутка экрана:
;              ----------------
Q10SCR  PROC    NEAR            ;AX уже установлен
       MOV     BH,1EH          ;Цвет желтый на синем
       MOV     CX,0000
       MOV     DX,184FH
       INT     10H             ;Прокрутка
       RET
Q10SCR  ENDP
;              Установка курсора:
;              -----------------
Q20CURS PROC    NEAR
       MOV     AH,02
       MOV     BH,00
       MOV     DL,00
       MOV     DH,ROW          ;Установить курсор
       INT     10H
       RET
Q20CURS ENDP
;              Вывод сообщения об ошибке на диске:
;              ----------------------------------
X10ERR  PROC    NEAR
       MOV     AH,09           ;DX содержит
       INT     21H             ; адрес сообщения
       MOV     ERRCDE,01       ;Установить код ошибки
       RET
X10ERR  ENDP

CODESG  ENDS
       END     BEGIN
```

Рис.16.1. Создание дискового файла.

Программа, приведенная на рис.16.1, создает дисковый файл по имени, которое вводится пользователем с клавиатуры. Блок FCB (FCBREC) в данной программе содержит следующие поля:

FCBDRIV Программа должна создать файл на диске в дисководе 4 (или D).
FCBNAME Имя файла - NAMEFILE.
FCBEXT Тип файла - DAT.
FCBBLK Начальное значение номера текущего блока - 0.
FCBRCSZ Размер записей неопределен, так как операция открытия устанавливает в данном поле значение 128.
FCBSQRC Начальное значение номера текущей записи - 0.

В программе организованы следующие процедуры:

BEGIN Инициализирует сегментные регистры, вызывает C10OPEN для создания файла и установки адреса DTA для DOS, вызывает D10PROC для ввода имени файла. Если ввод пустой, то происходит вызов G10PROC для завершения программы.
C10OPEN Создает для файла элемент в директории, устанавливает размер записей - 32 (шест.20) и инициализирует адрес буфера DTA для

DOS.
D10PROC Выдает запрос на ввод имен, вводит имена с клавиатуры и вызывает процедуру F10WRIT для записи вводимых имен на диск
E10DISP Управляет прокруткой и установкой курсора.
F10WRIT Записывает имена в дисковой файл.
G10CLSE Записывает маркер конца файла и закрывает файл.
X10ERR Выдает на экран сообщение об ошибке в случае не корректной операции создания файла или записи данных.

Каждая операция записи автоматически добавляет 1 к FCBSGRC (номер текущей записи) и шест.20 (размер записи) к FCBFLSZ (размер файла). Так как каждая запись имеет длину 32 байта, то операция заносит в буфер 16 записей и затем записывает весь буфер в сектор диска. Ниже показано содержимое DTA и буфера:

```
     DTA: |текущая запись|
   Буфер: |запись 00|запись 01|запись 02|...|запись 15|
```

Если пользователь ввел 25 имен, то счетчик записей увеличится от 1 до 25 (шест.19). Размер файла составит:

25 * 32 байта = 800 байтов или шест. 320

Операция закрытия заносит во второй сектор оставшиеся в буфере девять записей и изменяет в оглавлении дату и размер файла. Размер записывается байтами в переставленном порядке: 20030000. Последний буфер имеет следующий вид:

```
Буфер: |запись 16|запись 17|...|запись 24|шест.1A|...|...|
```

Для простоты в приведенной программе создаются записи файла, содержащие только одно поле. Записи большинства других файлов, однако, содержит различные символьные и двоичные поля и требуют описания записи в DTA. Если записи содержат двоичные числа, то не следует использовать маркер конца файла (EOF), так как двоичное число может совпасть с шест. кодом 1A.

Для того, чтобы сделать программу более гибкой, можно разрешить пользователю указать дисковод, на котором находится или будет находиться файл. В начале выполнения программа может выдать на экран сообщение, чтобы пользователь ввел номер дисковода, а затем изменить первый байт блока FCB.

ПОСЛЕДОВАТЕЛЬНОЕ ЧТЕНИЕ ДИСКОВОГО ФАЙЛА

В базовой версии DOS программа, читающая дисковый файл, содержит блок управления файлом, который определяет файл точно так, как он был создан. В начале программа для открытия файла использует функцию шест. OF:

```
MOV  AH,OFH        ;Открытие
LEA  DX,FCBname    ; файла
INT  21H           ;Вызов DOS
```

Операция открытия начинается с поиска в оглавлении элемента с именем и типом файла, определенными в FCB. Если такой элемент не будет найден в оглавлении, то в регистре AL устанавливается шест.FF. Если элемент найден, то в регистре AL устанавливается 00 и в FCB заносится действительный размер файла, а также устанавливается номер текущего блока в 0, длина записи в шест.80. После открытия можно заменить длину записи на другое значение.

DTA должно содержать определение считываемой записи в соответствии с

форматом, который использовался при создании файла. Для установки адреса DTA используется функция шест.1A (не путать с маркером конца файла EOF шест.1A) аналогично созданию дискового файла:

```
          MOV  AH,1AH         ;Установка
          LEA  DX,DTAname     ; адреса DTA
          INT  21H            ;Вызов DOS
```

Для последовательного чтения записей с диска используется функция шест.14:

```
          MOV  AH,14H         ;Последовательное
          LEA  DX,FCBname     ; чтение записей
          INT  21H            ;Вызов DOS
```

Чтение записи с диска по адресу DTA осуществляется на основе информации в блоке FCB. Операция чтения устанавливает в регистре AL следующие коды возврата:

00 Успешное чтение.
01 Конец файла, данные не прочитаны.
02 В DTA нет места для чтения одной записи.
03 Конец файла, прочитана частичная запись, заполненная нулями.

Первая операция чтения заносит содержимое всего сектора в буфер DOS. Затем операция определяет из блока FCB размер записи и пересылает первую запись из буфера в DTA. После дующие операции чтения пересылают остальные записи (если имеются) пока буфер не будет исчерпан. После этого операция чтения определяет адрес следующего сектора и заносит его содержимое в буфер.

После успешной операции чтения в   блоке   FCB   автоматически увеличивается номер текущей записи на 1. Завершение последовательного чтения определяется программой по маркеру конца файла (EOF), для чего в программе имеется соответствующая проверка. Так как оглавление при чтении файла не изменяется, то обычно нет необходимости закрывать файл после завершения чтения. Исключение составляют программы, которые открывают и читают несколько файлов одновременно. Такие программы должны закрывать файлы, так как DOS ограничивает число одновременно открытых файлов.

ПРОГРАММА: ИСПОЛЬЗОВАНИЕ FCB ДЛЯ ЧТЕНИЯ ДИСКОВОГО ФАЙЛА

```
TITLE   FCBREAD (EXE) Чтение записей созданных в CREATDSK
; -----------------------------------------------------
STACKSG SEGMENT PARA STACK 'Stack'
        DW      80 DUP(?)
STACKSG ENDS
;------------------------------------------------------
DATASG  SEGMENT PARA 'Data'
FCBREC  LABEL   BYTE             ;FCB для файла
FCBDRIV DB      04               ; дисковод D
FCBNAME DB      'NAMEFILE'       ; имя файла
FCBEXT  DB      'DAT'            ; тип файла
FCBBLK  DW      0000             ; номер текущего блока
FCBRCSZ DW      0000             ; длина логической записи
        DD      ?                ; размер файла (DOS)
        DW      ?                ; дата (DOS)
        DT      ?                ; зарезервировано (DOS)
```

```
FCBSQRC DB      00              ; текущий номер записи
        DD      ?               ; относительный номер

RECLEN  EQU     32              ;Длина записи
NAMEFLD DB      RECLEN DUP(' '), 13, 10, '$'

ENDCDE  DB      00
OPENMSG DB      '*** Open error ***', '$'
READMSG DB      '*** Read error ***', '$'
ROW     DB      00
DATASG  ENDS
;-------------------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
        ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASG
        PUSH    DS
        SUB     AX,AX
        PUSH    AX
        MOV     AX,DATASG
        MOV     DS,AX
        MOV     ES,AX
        MOV     AX,0600H
        CALL    Q10SCR          ;Очистить экран
        CALL    Q20CURS         ;Установить курсор
        CALL    E10OPEN         ;Открыть файл,
                                ; установить DTA
        CMP     ENDCDE,00       ;Открытие без ошибок?
        JNZ     A90             ; нет - завершить
A20LOOP:
        CALL    F10READ         ;Прочитать запись
        CMP     ENDCDE,00       ;Чтение без ошибок?
        JNZ     A90             ; нет - выйти
        CALL    G10DISP         ;Выдать имя на экран
        JMP     A20LOOP         ;Продолжить
A90:    RET                     ;Завершить
BEGIN   ENDP
;               Открытие файла на диске:
;               -----------------------
E10OPEN PROC    NEAR
        LEA     X,FCBREC
        MOV     AH,0FH          ;Функция открытия
        INT     21H
        CMP     AL,00           ;Файл найден?
        JNZ     E20             ; нет - ошибка

        MOV     FCBRCSZ,RECLEN  ;Длина записи (EQU)
        MOV     AH,1AH
        LEA     DX,NAMEFLD      ;Адрес DTA
        INT     21H
        RET
E20:
        MOV     ENDCDE,01       ;Сообщение об ошибке
        LEA     DX,OPENMSG
        CALL    X10ERROR
        RET
E10OPEN ENDP
;               Чтение дисковой записи:
;               ----------------------
F10READ PROC    NEAR
        MOV     AH,14H          ;Функция чтения
        LEA     DX,FCBREC
```

```
      INT     21H
      CMP     NAMEFLD,1AH     ;Считан маркер EOF?
      JNE     F20             ; нет
      MOV     ENDCDE,01       ; да
      JMP     F90
F20:
      CMP     AL,00           ;Чтение без ошибок?
      JZ      F90             ; да - выйти
      MOV     ENDCDE,01       ;Нет:
      CMP     AL,01           ;Конец файла?
      JZ      F90             ; да - выйти,
      LEA     DX,READMSG      ; нет - значит
      CALL    X10ERR          ; ошибка чтения
F90:
      RET
F10READ ENDP
;             Вывод записи на экран:
;             ---------------------
G10DISP PROC    NEAR
      MOV     AH,09           ;Функция вывода на экран
      LEA     DX,NAMEFLD
      INT     21H
      CMP     ROW,20          ;Последняя строка экрана?
      JAE     G30             ; нет...
      INC     ROW             ; да - увеличить строку
      JMP     G90
G30:
      MOV     AX,0601H
      CALL    Q10SCR          ; прокрутить
      CALL    Q20CURS         ; установить курсор
G90:  RET
G10DISP ENDP
;             Прокрутка (скроллинг) экрана:
;             ----------------------------
Q10SCR  PROC    NEAR          ;AX уже установлен
      MOV     BH,1EH          ;Установить цвет
      MOV     CX,0000
      MOV     DX,184FH        ;Функция прокрутки
      INT     10H
      RET
Q1OSCR  ENDP
;             Установка курсора:
;             -----------------
Q20CURS PROC    NEAR
      MOV     AH,02
      MOV     BH,00
      MOV     DH,ROW
      MOV     DL,00
      INT     10H
      RET
Q20CURS ENDP
;             Вывод сообщения об ошибке на диске:
;             ----------------------------------
X10ERR  PROC    NEAR
      MOV     AH,09           ;DX содержит адрес
      INT     21H             ; сообщения
      RET
X10ERR  ENDP

CODESG  ENDS
      END     BEGIN
```

Рис.16.2. Чтение дискового файла

На рис.16.2 приведена программа, которая выполняет чтение файла, созданного предыдущей программой, и вывод на экран имен из записей файла. Обе программы содержат идентичные блоки FCB, хотя, имена полей FCB могут быть различны. Содержимое полей имени и типа файла должны быть одинаковы.

Программа содержит следующие процедуры:

BEGIN Инициализирует сегментны регистра, вызывает процедуру E10OPEN для открытия файла и установки DTA и вызывает F10READ для чтения записей. Если считан маркер конца файла, то программа завершается, если нет, то вызывается процедура G10DISP.
E10OPEN Открывает файл, устанавливает значение размера и записей, равное 32 (шест.20), и инициализирует адрес DTA.
F10READ Выполняет последовательное чтение записей. Операция чтения автоматически увеличивает номер текущей записи в блоке FCB.
G10DISP Выводит на экран содержимое прочитанной записи.
X10ERR Выводит на экран сообщение об ошибке в случае некорректной операции открытия или чтения.

Операция открытия выполняет поиск имени и типа файла в оглавлении. Если необходимый элемент оглавления найден, то автоматически в блок FCB заносятся размер файла, дата и длина записей. Первая операция чтения записи с номером 00 получает доступ к диску и считывает весь сектор (16 записей) в буфер. После этого первая запись заносится в DTA, а номер текущей записи в FCB увеличивается с 00 до 01:

```
Буфер: |запись 00|запись 01|запись 02|... |запись 15|
         DTA : |запись 00|
```

Второй операции чтения нет необходимого обращаться к диску. Так как требуемая запись уже находится в буфере, то операция просто пересылает запись 01 из буфера в DTA и увеличивает номер текущей записи на единицу. Таким же образом выполняются следующие операции чтения пока все 16 записей из буфера не будут обработаны.

Операции чтения 16-ой записи приводит к физическому чтению следующего сектора в буфер и пересылка первой записи сектора в DTA. Последующие операции чтения переносят остальные записи из буфера в DTA. Попытка прочитать после последней записи вызовет состояние конца файла и в регистр AL будет записан код возврата шест.01.

ПРЯМОЙ ДОСТУП

До сих пор в этой главе рассматривалась последовательная обработка дисковых файлов, которая адекватна как для создания файла, так и для печати его содержимого или внесения изменений в небольшие файлы. Если программа ограничена только возможностью последовательной обработки, то для изменения файла она должна считывать каждую запись, вносить изменения в определенные из них и заносить записи в другой файл (программа может использовать один DTA, но потребуются различные блоки FCB). Обычной практикой является чтение входного файла с диска A и запись обновленного файла на диск B. Преимущество этого способа состоит в том, что он автоматически оставляет резервную копию.

В некоторых случаях применяется доступ к конкретным записям файла для получения информации, например, нескольких служащих или о части ассортимента товаров. Для доступа, скажем, к 300-ой записи файла,

последовательная обработка должна включать чтение всех 299 предшествующих записей, пока не будет получена 300-я запись. П р и м е ч а н и е: система может начать обработку с конкретного номера блока и записи).

Несмотря на то, что файл создается последовательно, доступ к записям может быть последовательным или прямым (произвольным). Требования прямой обработки, использующей вызов DOS, заключаются в установке требуемого номера записи в соответствующее поле FCB и выдаче команды прямого чтения или записи.

Произвольный доступ использует относительный номер записи (байты 33-36) в блоке FCB. Поле имеет размер двойного слова и использует обратную последовательность байт в словах. Для локализации требуемой записи система автоматически преобразует относительный номер записи в номер текущего блока (байты 12-13) и номер текущей записи (байт 32).

Прямое чтение

Операции открытия и установки DTA одинаковы как для прямой, так и для последовательной обработки. Предположим, что программа должна выполнить прямой доступ к пятой записи файла. Установим значение 05 в поле FCB для относительного номера записи и выполним команды для прямого чтения. В результате успешной операции содержимое пятой записи будет помещено в DTA.

Для прямого чтения записи необходимо поместить требуемое значение относительного номера записи в FCB и вызвать функцию шест.21:

```
MOV  AH,21H         ;Запрос на
LEA  DX,FCBname     ; прямое чтение
INT  21H            ;Вызов DOS
```

Операция чтения преобразует относительный номер записи в номера текущего блока и записи. Полученные значения используются для локализации требуемой дисковой записи, передачи содержимого записи в DTA и установки в регистр AL следующие значения:

00 Успешное завершение
01 Данные не доступны
02 Чтение прекращено из-за нехватки места в DTA
03 Прочитана частичная запись, заполненная нулями.

Как видно, среди перечисленных кодов возврата отсутствует состояние конец файла. При корректном чтении записи предполагается единственный код возврата - 00. Остальные коды возврата могут являться результатом установки неправильного относительного номера записи или некорректная установка адреса DTA или FCB. Так как такие ошибки легко допустить, то полезно выполнять проверку регистра AL на ненулевое значение.

Когда программа выдает первый запрос на прямую запись, операция, используя оглавление для локализации сектора, на котором находится требуемая запись, считывает весь сектор с диска в буфер и пересылает запись в DTA. Предположим, например, что записи имеют размер 128 байт, т.е. четыре записи в одном секторе. Запрос на прямое чтение записи 23 приводит к чтению в буфер четырех записей, лежащих в одном секторе:

```
| запись 20 | запись 21 | запись 22 | запись 23 |
```

Когда программа вновь выдаст прямой запрос на запись, например, 23, то операция сначала проверит содержимое буфера. Так как данная запись уже находится в буфере, то она непосредственно пересылается в DTA. Если программа запросит запись 35, который нет в буфере, операция через оглавление локализует требуемую запись, считает весь сектор в буфер и поместит запись в DTA. Таким образом, операции прямого доступа к записям более эффективны, если номера записей близки друг к другу.

Прямая запись

Операция создания файла и установки DTA одинаковы как для прямого, так и для последовательного доступа. Для обработки файла учета товаров программа может, используя прямой доступ, считать необходимую запись, внести, введенные вручную, изменения (например, новое количество товаров) и вернуть запись на диск на то же место. Операция прямой записи использует относительный номер записи в блоке FCB и функцию шест.22 следующим образом:

```
MOV  AH,22H       ;Запрос на
LEA  DX,FCBname    ; прямую запись
INT  21H          ;Вызов DOS
```

Операция устанавливает в регистре AL следующие коды возврата:

00 Успешная операция
01 На диске нет места
02 Операция прекращена в результате недостаточного места в DTA.

При создании нового файла прямым доступом может быть получен ненулевой код возврата. Но при прямом чтении и переписывании измененных записей на том же месте диска код возврата должен быть только 00.

Относительный номер записи в блоке FCB при прямом доступе имеет размер двойного слова (четыре байта), каждое слово записывается обратной последовательностью байтов. Для небольших файлов возможно потребуется установка лишь самого левого байта или слова, но для больших файлов установка номера записи в трех или в четырех байтах требует некоторой тщательности.

ПРОГРАММА: ПРЯМОЕ ЧТЕНИЕ ДИСКОВОГО ФАЙЛА

```
      page    60,132
TITLE   RANREAD (COM) Прямое чтение записей,
;                     созданных в FCBCREAT
CODESG  SEGMENT PARA 'Code'
      ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:CODESG
      ORG     100H
BEGIN:  JMP     MAIN
;----------------------------------------------------
FCBREC  LABEL   BYTE            ;FCB для дискового файла
FCBDRIV DB      04              ; дисковод D
FCBNAME DB      'MAMEFILE'      ; имя файла
FCBEXT  DB      'DAT'           ; тип файла
FCBBLK  DW      0000            ; номер текущего блока
FCBRCSZ DW      0000            ; длина логической записи
      DD      ?               ; размер файла (DOS)
      DW      ?               ; дата (DOS)
      DT      ?               ; зарезервировано (DOS)
      DB      00              ; номер текущей записи
FCBRNRC DD      000000000       ; относительный номер

RECLEN  EQU     32              ;Длина записи
RECDPAR LABEL   BYTE            ;Список параметров:
MAXLEN  DB      3               ;
ACTLEN  DB      ?               ;
RECDNO  DB      3 DUP(' ')      ;
```

```
NAMEFLD DB      RECLEN DUP(' '),13,10,'$' ;DTA

OPENMSG DB      '*** Open error ***',13,10,'$'
READMSG DB      '*** Read error ***',13,10,'$'
COL     DB      00
PROMPT  DB      'Record number? $'
ROW     DB      00
ENDCDE  DB      00
;-----------------------------------------------------
MAIN    PROC    NEAR
        CALL    Q10CLR          ;Очистить экран
        CALL    Q20CURS         ;Установить курсор
        CALL    C10OPEN         ;Открыть файл,
                         ; установить DTA
        CMP     ENDCDE,00       ;Открытие без ошибок?
        JZ      A20LOOP         ; да - продолжить,
        RET                     ; нет - завершить
A20LOOP:
        CALL    D10RECN         ;Получить номер записи
        CMP     ACTLEN,00       ;Есть запрос?
        JE      A40             ; нет - выйти
        CALL    F10READ         ;Чтение (прямой доступ)
        CMP     ENDCDE,00       ;Есть ошибки чтения?
        JNZ     A30             ; да - обойти
        CALL    G10DISP         ;Вывести на экран
A30:
        JMP     A20LOOP
A40:    RET                     ;Завершить программу
MAIN    ENDP
;               Подпрограмма открытия файла на диске:
;               -----------------------------------
C10OPEN PROC    NEAR
        MOV     AH,0FH          ;Функция открытия
        LEA     DX,FCBREC
        INT     21H
        CMP     AL,00           ;Открытие нормальное?
        JNZ     C20             ; нет - ошибка
        MOV     FCBRCSZ,RECLEN  ;Длина записи (EQU)
        MOV     AH,1AH
        LEA     DX,NAMEFLD      ;Установить адрес DTA
        INT     21H
        RET
C20:
        LEA     DX,OPENMSG
        CALL    X10ERR
        RET
C10OPEN ENDP
;               Ввод с клавиатуры номера записи:
;               ------------------------------
D10RECN PROC    NEAR
        MOV     AH,09H          ;Функция вывода на экран
        LEA     DX,PROMPT
        INT     21H
        MOV     AH,0AH          ;Функция ввода с клавиатуры
        LEA     DX,RECDPAR
        INT     21H
        CMP     ACTLEN,01       ;Проверить длину (0,1,2)
        JB      D40             ;Длина 0, завершить
        JA      D20
        SUB     AH,AH           ;Длина 1
```

```
       MOV     AL,RECDNO
       JMP     D30
D20:
       MOV     AH,RECDNO       ;Длина 2
       MOV     AL,RECDNO+1
D30:
       AND     AX,0F0FH        ;Удалить ASCII тройки
       AAD                     ;Преобразовать в двоичное
       MOV     WORD PTR FCBRNRC,AX
D40:
       MOV     COL,20
       CALL    Q20CURS         ;Установить курсор
       RET
D10RECN ENDP
;              Чтение дисковой записи:
;              ----------------------
F10READ PROC    NEAR
       MOV     ENDCDE,00       ;Очистить код завершения
       MOV     AH,21H          ;Функция прямого чтения
       LEA     DX,FCBREC
       INT     21H
       CMP     AL,00           ;Чтение без ошибок?
       JZ      F20             ; да - выйти
       LEA     DX,READMSG      ; нет - выдать
       CALL    X10ERR          ; сообщение об ошибке
F20:   RET
F10READ ENDP
;              Вывод имени на экран:
;              --------------------
G10DISP PROC    NEAR
       MOV     AH,09           ;Функция вывода на экран
       LEA     DX,NAMEFLD
       INC     21H
       INC     ROW
       MOV     COL,00
       RET
G10DISP ENDP
;              Очистка экрана:
;              --------------
Q10CLR  PROC    NEAR
       MOV     AX,0600H        ;Функция прокрутки
       MOV     BH,41H          ;Цвет (07 для ч/б)
       MOV     CX,0000
       MOV     DX,184FH
       INT     10H
       RET
Q10CLR  ENDP
;              Установка курсора:
;              -----------------
Q20CURS PROC    NEAR
       MOV     AH,02           ;Функция установки
       MOV     BH,00           ; курсора
       MOV     DH,ROW
       MOV     DL,COL
       INT     10H
       RET
Q20CURS ENDP
;              Вывод сообщения об ошибке на диске:
;              ----------------------------------
X10ERR  PROC    NEAR
       MOV     AH,09           ;DX содержит адрес
```

```
        INT     21H             ; сообщения
        INC     ROW
        MOV     ENDCDE,01
        RET
X10ERR  ENDP
        CODESG  ENDS
        END     BEGIN
```

Рис.16.3. Прямое чтение дисковых записей.

На рис.16.3 приведена программа, которая считывает файл, созданный предыдущей программой (см. рис.16.1). Вводя любой относительный номер записи, лежащей в границах файла, пользователь запрашивает вывод на экран любой записи файла. Если файл содержит 25 записей, то правильными номера являются номера от 00 до 24. Номер вводится с клавиатуры в ASCII формате и должен быть в нашем случае одно- или двузначным числом.

Программа содержит следующие процедуры:

C10OPEN Открывает файл, устанавливает размер записи 32 и устанавливает адрес DTA.
D10RECN Вводит номер записи с клавиатуры, преобразует его в двоичный формат и записывает полученное значение в FCB. В качестве усовершенствования процедуры можно вставить проверку вхождения номера в границы от 00 до 24.
F10READ Помещает требуемую запись в DTA в соответствии с относительным номером записи в FCB.
G10DISP Выводит запись на экран.

Процедура D10RECN вводит номер записи с клавиатуры и проверяет длину ввода в списке параметров. Возможны три варианта:

00 Запрошен конец обработки
01 Введено однозначное число (в регистре AL)
02 Введено двузначное число (в регистре AX)

Данная процедура преобразует введенное число из ASCII формата в двоичный формат. Так как значение находится в регистре AX, то лучше использовать команду AAD для преобразования. После преобразования двоичный код из регистра AX пересылается в два левых байта поля относительного номера записи в блоке FCB. Если, например, введено число 12 в ASCII формате, то AX будет содержать 3132. Команда AND преобразует это значение в 0102, а команда AAD - в 000C. Результат преобразования заносится в поле относительного номера записи блока FCB в виде C000 0000.

ПРЯМОЙ БЛОЧНЫЙ ДОСТУП

Если в программе имеется достаточно места, то одна прямая блочная операция может записать весь файл из DTA на диск, а также прочитать весь файл с диска в DTA. Данная особенность весьма полезна для записи на диск таблиц, которые другие программы могут считывать в память для обработки.

Начать можно с любого правильного относительного номера записи. Число записей также может быть любым, хотя блок должен находится в пределах файла. Перед началом необходимо открыть файл и инициализировать DTA.

Для операции прямой блочной записи необходимо установить в регистре CX требуемое число записей, установить в FCB стартовый относительный номер записи и выдать функцию шест.28:

```
        MOV  AH,28H         ;Операция прямой блочной записи
        MOV  CX,records     ;Установка числа записей
        LEA  DX,FCBname     ;
        INT  21H            ;Вызов DOS
```

Операция преобразует относительный номер записи в текущие номер блока и номер записи. Полученные значения используются для определения начального адреса на диске. В результате операции в регистре AL устанавливаются следующие коды возврата:

```
        00   Успешное завершение для всех записей
        01   На диске недостаточно места.
```

Кроме того операция устанавливает в FCB в поле относительного номера записи и полях текущих номеров блока и записи значения, соответствующие следующему номеру записи. Например, если были записаны записи с 00 до 24, то следующий номер записи будет 25 (шест.19).

Для операции прямого блочного чтения необходимо установить в регистре CX требуемое число записей и использовать функцию шест.27:

```
        MOV  AH,27H         ;Операция прямого блочного чтения
        MOV  CX,records     ;Установка числа записей
        LEA  DX,FCBname     ;
        INT  21H            ;Вызов DOS
```

Операция чтения возвращает в регистре AL следующие значения:

```
        00   Успешное чтение всех записей
        01   Прочитана последняя запись файла
        02   Прочитано предельное для DTA число записей
        03   Прочитана последняя запись файла не полностью.
```

В регистре CX остается действительное число прочитанных записей, а в FCB в поле относительного номера записи и полях текущих номеров блока и записи устанавливаются значения, соответствующие следующему номеру записи.

Если необходимо загрузить в память весь файл, но число записей неизвестно, то следует после операции открытия разделить размер файла на длину записи. Например, для размера файла шест.320 (800) и длине записи шест.20 (32) число записей будет шест.19 (25).

ПРОГРАММА: ПРЯМОЕ БЛОЧНОЕ ЧТЕНИЕ

```
TITLE   RANBLOK (COM) Прямое блочное чтение файла
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:CODESG
        ORG     100H
BEGIN:  JMP     MAIN
;-------------------------------------------------
FCBREC  LABEL   BYTE            ;FCB для дискового файла
FCBDRIV DB      04              ; дисковод D
FCBNAME DB      'NAMEFILE'      ; имя файла
FCBEXT  DB      'DAT'           ; тип файла
FCBBLK  DW      0000            ; номер текущего блока
FCBRCSZ DW      0000            ; логическая длина записи
FCBFLZ  DD      ?               ; DOS размер файла
        DW      ?               ; DOS дата
        DT      ?               ; DOS зарезервировано
```

```
        DB      00              ; номер текущей записи
FCBRNRC DD      00000000        ; относительный номер

DSKRECS DB      1024 DUP(?),'$' ;DTA для блока записей

ENDCODE DB      00
NORECS  DW      25              ;Число записей
OPENMSG DB      '*** Open error ***',13,10,'$'
READMSG DB      '*** Open error ***',13,10,'$'
ROWCTR  DB      00
; ----------------------------------------------------
MAIN    PROC    NEAR
        CALL    Q10CLR          ;Очистить экран
        CALL    Q20CURS         ;Установить курсор
        CALL    E10OPEN         ;Открыть файл,
                                ; установить DTA
        CMP     ENDCODE,00      ;Успешное открытие?
        JNZ     A30             ; нет - выйти
        CALL    F10READ         ;Читать записи
        CALL    G10DISP         ;Вывод блока на экран
A30:    RET                     ;Завершить программу
MAIN    ENDP
;               Открыть дисковый файл:
;               ---------------------
E10OPEN PROC    NEAR
        MOV     AH,0FH          ;Функция открытия файла
        LEA     DX,FCBREC
        INT     21H
        CMP     AL,00           ;Успешное открытие?
        JNZ     A30             ; нет - ошибка

        MOV     FCBRCSZ,0020H   ;Размер записи
        MOV     AH,1AH
        LEA     DX,DSKRECS      ;Установить адрес DTA
        INT     21H
        RET
E20:
        LEA     DX,OPENMSG      ;Ошибка открытия файла
        CALL    X10ERR
        RET
E10OPEN ENDP
;               Чтение блока:
;               ------------
F10READ PROC    NEAR
        MOV     AH,27H          ;Прямое чтение блока
        MOV     CX,NORECS       ;Число записей
        LEA     DX,FCBREC
        INT     21H
        MOV     ENDCODE,AL      ;Сохранить код возврата
        RET
F10READ ENDP
;               Вывод блока на экран:
;               --------------------
G10DISP PROC    NEAR
        MOV     AH,09           ;Функция вывода на экран
        LEA     DX,DSKRECS
        INT     21H
        RET
G10DISP ENDP
;               Подпрограмма очистки экрана:
;               --------------------------
```

```
Q10CLR  PROC    NEAR
        MOV     AX,0600H        ;Функция скроллинга
        MOV     BH,41H          ;Цвет (07 для ч/б)
        MOV     CX,0000
        MOV     DX,184FH
        INT     10H
        RET
Q10CLR  ENDP
;               Подпрограмма установки курсора:
;               -----------------------------
Q20CURS PROC    NEAR
        MOV     AH,02           ;Функция установки курсора
        MOV     BH,00
        MOV     DH,ROWCTR
        MOV     DL,00
        INT     10H
        INC     ROWCTR
        RET
Q20CURS ENDP
;               Подпрограмма сообщения об ошибке диска:
;               -------------------------------------
X10ERR  PROC    NEAR
        MOV     AH,09           ;DX содержит адрес
        INT     21H             ; сообщения
        MOV     ENDCODE,01
        RET
X10ERR  ENDP

CODESG  ENDS
        END     BEGIN
```

Рис.16.4. Прямое блочное чтение.

На рис.16.4 приведена программа, выполняющая блочное чтение файла, созданного программой на рис.16.1. Программа устанавливает начальный относительный номер записи 00, в регистре CX - счетчик на 25 записей и выводит на экран всю информацию из DTA (только для того, чтобы убедиться, что информация считана). Другие варианты программы могут включать установку другого начального номера записи и считывание менее 25 записей.

В программе организованы следующие процедуры:

E10OPEN Открывает файл, устанавливает размер записи в FCB равным 32 и устанавливает адрес DTA.
F10READ Устанавливает число записей равным 25 и выполняет блочное чтение
G10DISP Выводит блок на экран.

Операция чтения преобразует относительный номер записи 00 в FCB в номер текущего блока 00 и номер текущей записи 00. В конце операции чтения в FCB текущий номер записи будет содержать шест.19, а относительный номер записи - шест.19000000.

АБСОЛЮТНЫЕ ОПЕРАЦИИ ДИСКОВОГО ВВОДА-ВЫВОДА

Для непосредственного доступа к диску можно использовать операции абсолютного чтения и абсолютной записи с помощью функций DOS INT 25H и 26H. В этом случае не используются оглавление диска и преимущества блокирования и разблокирования записей, обеспечиваемые функцией DOS INT

21H.

Абсолютные операции предполагают, что все записи имеют размер сектора, поэтому прямой доступ осуществляется к полному сектору или блоку секторов. Адресация диска выполняется по "логическому номеру записи" (абсолютный сектор). Для определения логического номера записи на двухсторонних дискетах с девятью секторами счет секторов ведется с дорожки 0, сектора 1, следующим образом:

Дорожка	Сектор	Логический номер записи
0	1	0
0	2	1
1	1	9
1	9	17
2	9	26

Для двухсторонних дискет используется следующая формула:

Логический номер записи = (дорожка x 9) + (сектор - 1)

Например, логический номер записи на дорожке 2 и секторе 9 определяется как

(2 x 9) + (9 - 1) = 18 + 8 = 26

Фрагмент программы для абсолютных операций ввода-вывода:

```
MOV  AL,drive#      ;0 для A, 1 для B и т.д.
MOV  BX,addr        ;Адрес области ввода-вывода
MOV  CX,sectors     ;Число секторов
MOV  DX,record#     ;Начальный логический номер записи
INT  25H или 26H    ;Абсолютное чтение или запись
```

Операции абсолютного чтения или запись разрушают содержимое всех регистров, кроме сегментных, и устанавливают флаг CF для индикации успешной (0) или безуспешной (1) операции. В случае безуспешной операции содержимое регистра AL описывает характер ошибки:

AL	Причина
1000 0000	Устройство не отвечает
0100 0000	Ошибка установки головок
0010 0000	Ошибка контролера
0001 0000	Ошибка дискеты?
0000 1000	Переполнение DMA при чтении
0000 0100	Сектор не найден
0000 0011	Попытка записи на защищенной дискете
0000 0010	Не найден адресный маркер

Команда INT записывает содержимое флагового регистра в стек. После завершения команды INT следует восстановить флаги, но проверив перед этим флаг CF.

ДРУГИЕ ДИСКОВЫЕ ОПЕРАЦИИ

Кроме основных дисковых функций DOS имеется несколько дополнительных полезных дисковых операций.

Сброс диска: Шест.D

Обычно нормальное закрытие файла приводит к занесению всех оставшихся в буфере записей на диск и корректировке оглавления. В особых случаях (между шагами программы или аварийном завершении) может потребоваться сброс диска. Функция DOS шест.D освобождает все файловые буфера и не корректирует оглавление диска. Если необходимо, то вначале данная функция закрывает все файлы.

```
MOV  AH,ODH    ;Запрос на сброс диска
INT  21H       ;Вызов DOS
```

Установка текущего дисковода: Шест.E

Основное назначение функции DOS шест.E - установка номера текущего (по умолчанию) дисковода. Номер дисковода помещается в регистр DL, причем 0 соответствует дисководу A, 1 - B и т.д.

```
MOV  AH,OEH    ;Запрос на установку
MOV  DL,02     ; дисковода C
INT  21H       ;Вызов DOS
```

Операция возвращает в регистр AL число дисководов (независимо от типа). Так как для DOS необходимо по крайней мере 2 логических дисковода A и B, то DOS возвращает значение 02 и для систем с одним дисководом. (Для определения действительного числа дисководов используется команда INT 11H).

Поиск элементов оглавления: шест.11 и 12

Программной утилите может потребоваться поиск в оглавлении для доступа к имени файла, например, при удалении или переименовании. Для доступа к первому или единственному элементу оглавления необходимо загрузить в регистр DX адрес неоткрытого блока FCB и выполнить функцию 11H. При использовании расширенного блока FCB можно также получить код атрибута (см.техническое руководство по DOS).

```
MOV  AH,11H         ;Запрос на первый элемент
LEA  DX,FCBname     ;Неоткрытый FCB
INT  21H            ;Вызов DOS
```

FCB может быть расположено по адресу 5CH в префиксе программного сегмента, предшествующем программе в памяти (DTA по умолчанию). Подробно см. гл.22.

В регистре AL операция возвращает шест.FF, если элемент не найден, и шест.00, если найден. Операция устанавливает в DTA номер дисковода (1=A, 2=B и т.д.) имя файла и тип файла.

Если найдено несколько элементов при выборке по шаблону (например, *.ASM), то для локализации элементов подмножества директории используется функция 12H:

```
MOV  AH,12H         ;Запрос следующего элемента
LEA  DX,FCBname     ;Неоткрытый FCB
INT  21H            ;Вызов DOS
```

Коды возврата в регистре AL аналогичны кодам функции 11H.

Удаление файла: шест.13

Для удаления файла в программе используется функция DOS 13H. Операция удаления устанавливает специальный байт в первой позиции имени файла в оглавлении.

```
MOV  AH,13H        ;Запрос на удаление файла
LEA  DX,FCBname     ;Неоткрытый FCB
INT  21H           ;Вызов DOS
```

Если операция находит и удаляет элемент, то в регистре AL устанавливается код возврата 00, иначе код равен шест.FF.

Переименование файла: шест.17

Для переименования файла в программе используется функция DOS шест.17. Старое имя файла записывается в обычном месте блока FCB, а новое - начиная со смещения 16.

```
MOV  AH,17H        ;Запрос на переименование
LEA  DX,FCBname     ;Адрес FCB
INT  21H           ;Вызов DOS
```

Символы ? и * в новом имени приводят к сохранению в соответствующих позициях символов из старого имени. Успешная операция устанавливает в регистре AL код возврата 00, а безуспешная (файл по старому имени не найден или по новому имени уже существует) - код FF.

Получение текущего номера дисковода: шест.19

Функция DOS шест.19 позволяет определить текущий номер дисковода:

```
MOV  AH,19H        ;Получить текущий дисковод
INT  21H           ;Вызов DOS
```

Операция возвращает шест. номер дисковода в регистре AL (0=A, 1=B и т.д.). Полученное значение можно поместить непосредственно в FCB для доступа к файлу с текущего дисковода.

Кроме перечисленных существуют функции для получения информации из таблицы FAT (1B и 10), установки поля прямой записи (24), установки вектора прерываний (25), создания нового программного сегмента (26) и анализа имени файла (29). Эти функции описаны в техническом руководстве по DOS.

ПРОГРАММА: ВЫБОРОЧНОЕ УДАЛЕНИЕ ФАЙЛОВ

```
TITLE   SELDEL (COM)      Выборочное удаление файлов
;              Предполагается текущий дисковод;
;              Примеры параметров: *.*, *.BAK, и т.д.
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
        ORG     100H
BEGIN   JMP     MAIN
; --------------------------------------------------------
TAB     EQU     09
LF      EQU     10
```

```
CR      EQU     13
CRLF    DB      CR,LF,'$'
DELMSG  DB      TAB,'Erase','$'
ENDMSG  DB      CR,LF,'No more directory entries',CR,LF,'$'
ERRMSG  DB      'Write protected disk','$'
PROMPT  DB      'y = Erase, N = Keep, Ret = Exit',CR,LF,'$'
; -----------------------------------------------------------
MAIN    PROC    NEAR            ;Главная процедура
        MOV     AH,11H          ;Найти первый элемент
        CALL    D10DISK
        CMP     AL,0FFH         ;Если нет элементов,
        JE      A90             ; то выйти
        LEA     DX,PROMPT       ;Текст запроса
        CALL    B10DISP
A20:
        LEA     DX,DELMSG       ;Выдать сообщение
        CALL    B10DISP         ; об удалении файла
        MOV     CX,11           ;11 символов
        MOV     SI,81H          ;Начало имени файла
A30:
        MOV     DL,[SI]         ;Текущий символ
        CALL    C10CHAR         ; для вывода на экран
        INC     SI              ;Следующий символ
        LOOP    A30
        MOV     DL,'?'
        CALL    C10CHAR
        MOV     AH,01           ;Получить односимвольный
        INT     21H             ; ответ
        CMP     AL,ODH          ;Символ Return?
        JE      A90             ; да - выйти
        OR      AL,00100000B    ;Перекодировать
                                ; в прописную букву
        CMP     AL,'y'          ;Запрошено удаление?
        JNE     A50             ; нет - обойти,
        MOV     AH,13H          ; да - удалить файл
        MOV     DX,80H
        INT     21H
        CMP     AL,0            ;Успешное удаление?
        JZ      A50             ; да - обойти
        LEA     DX,ERRMSG       ; нет - выдать
        CALL    B10DISP         ; предупреждение
        JMP     A90
A50:
        LEA     DX,CRLF         ;Перевести строку на экране
        CALL    B10DISP
        MOV     AH,12H
        CALL    B10DISK         ;Получить следующий элемент
        CMP     AL,0FFH         ;Есть еще?
        JNE     A20             ; да - повторить
A90:
        RET                     ;Выход в DOS
MAIN    ENDP
;               Вывод строки на экран;
;               ---------------------
B10DISP PROC    NEAR            ;в DX находится адрес
        MOV     AH,09           ; строки
        INT     21H
        RET
B10DISP ENDP
;               Вывод символа на экран;
;               ----------------------
```

```
C10CHAR PROC    NEAR            ;в DL находится символ
       MOV     AH,02
       INT     21H
       RET
C10CHAR ENDP
;              Чтение элемента каталога:
;              ------------------------
D10DISK PROC    NEAR
       MOV     DX,5CH         ;Установить FCB
       INT     21H
       CMP     AL,0FFH        ;Есть еще элементы?
       JNE     D90
       PUSH    AX             ;Сохранить AL
       LEA     DX,ENDMSG
       CALL    B10DISP
       POP     AX             ;Восстановить AL
D90:    RET
D10DISK ENDP

CODESG  ENDS
       END     BEGIN
```

Рис.16.5. Выборочное удаление файлов.

На рис.16.5 приведена COM-программа по имени SDEL, иллюстрирующая функции DOS 11H, 12H и 13H для удаления выбранных файлов. Для запроса на удаление файлов пользователь может ввести,например, следующие команды:

```
SDEL  *.*      (все файлы)
SDEL  *.BAK    (все BAK-файлы)
SDEL  TEST.*   (все файлы по имени TEST)
```

Посредством DOS программа определяет в оглавлении элементы, удовлетворяющие запросу. DOS заносит полное имя найденного элемента в PSP (префикс программного сегмента) по смещению шест.81 (DTA по умолчанию). Затем программа выводит на экран имя файла и запрос подтверждения. Ответ Y (да) разрешает удаление, N (нет) сохраняет файл, а Return завершает выполнение.

Обратите внимание на то, что данная программа должна быть создана как COM-программа, так как EXE-программа требуют отличной адресации для использования смещений шест.5C и 81 в PSP. Для тестирования программы используйте скопированные временно файлы.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Программа, использующая INT 21H в базовой версии DOS для операций ввода-вывода на диск, должна содержать блок управления файлом (FCB) для каждого доступного файла.

- Один блок содержит 128 записей. Номер текущего блока и номер текущей записи в FCB указывают на дисковую запись, которая должна быть обработана.

- В обратной последовательности байт в FCB записываются следующие элементы: номер текущего блока, размер записи, размер файла и относительный номер записи.

- Все программы, обрабатывающие один и тот же файл, должны иметь одинаково описанный блок FCB.

- Область ввода-вывода (DTA) определяется адресом памяти, куда должна быть помещена запись при чтении или откуда она заносится на диск. Прежде, чем выполнить операцию записи или чтения, в программе необходимо установить каждую область DTA.

- Операция открытия файла устанавливает в блоке FCB значения для следующих элементов: имя файла, тип файла, размер записи (шест.80), размер файла и дата. Программа должна заменить размер записей на правильное значение.

- Программа, использующая для записи файла операцию DOS INT 21H, должна закрыть файл в конце обработки для того, чтобы поместить на диск все оставшиеся в буфере записи (если таковые имеются) и скорректировать соответствующий элемент оглавления.

- При использовании для чтения и записи операции DOS INT 21H система автоматически изменяет текущий номер записи в FCB.

- Операция чтения по прерыванию DOS INT 21H проверяет наличие требуемой записи сначала в буфере и при отсутствии выполняет чтение с диска.

- Прямой метод доступа требует указания номера записи в поле относительного номера записи блока FCB.

- Восемь байт (двойное слово) относительного номера записи кодируются в обратной последовательности байт.

- Если требуемая запись при прямом доступе уже находится в буфере, то система передает ее непосредственно в DTA. В противном случае выполняется чтение с диска в буфер всего сектора, содержащего необходимую запись.

- Операции прямого блочного чтения и записи более эффективны при наличии достаточной памяти. Эти операции особенно удобны для загрузки таблиц.

- Команды DOS INT 25H и 26H осуществляют дисковые операции абсолютного чтения и записи, но не поддерживают обработку оглавления, не определяют конец файла и не обеспечивают блокирование и деблокирование записей.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

16.1. Напишите функции базовой версии DOS для следующих операций: а) создание файла, б) установка DTA, в) последовательная запись, г) открытие файла, д) последовательное чтение.

16.2. Программа использует размер записи, устанавливаемый при открытии файла по умолчанию. а) Сколько записей содержит один сектор? б) Сколько записей содержит дискета с тремя дорожками по девять секторов на каждой? в) Если на дискете (б) находится один файл, то при последовательном чтении сколько произойдет физических обращений к диску?

16.3. Напишите программу, которая создает дисковый файл, содержащий записи из трех элементов: номер товара (пять символов), наименование товара (12 символов) и стоимость единицы товара (одно слово). Ввод этих

значений должен осуществляться пользователем с клавиатуры. Не забудьте преобразовать числа из ASCII представления в двоичное представление.

16.4. Напишите программу, которая выводит на экран файл, созданный в вопросе 16.3.

16.5. Определите текущий блок и запись для следующих номеров записей при прямом доступе: а)45, б)73, в)150, г)260.

16.6. В каком виде номер записи 2652 (десятичное) устанавливается в поле относительной записи блока FCB?

16.7. Укажите шестнадцатеричные номера функций для следующих операций: а) прямая запись, б) прямое чтение, в) прямая блочная запись, г) прямое блочное чтение.

16.8. Напишите команды для определения числа записей файла, предполагая, что операция открытия уже выполнена. Имена полей с размером файла FCB FLSZ и размером записи FCB FCSZ.

16.9. Используя программу из вопроса 16.4 для создания файла с количеством, ценами и наименованиями товаров, сформируйте файл с приведенными ниже данными. Напишите программу, которая выполняет одно блочное чтение данного файла и выводит каждую запись на экран.

Номер	Цена	Наименование
023	00315	Ассемблеры
024	00430	Компоновщики
027	00525	Компиляторы
049	00920	Компрессоры
114	11250	Экстракторы
117	00630	Буксиры
122	10520	Лифты
124	21335	Процессоры
127	00960	Станки для наклеивания меток
232	05635	Черпатели?
999	00000	

16.10. Измените программу из вопроса 16.9 так, чтобы цены записывались на диск в двоичном формате.

16.11. Измените программу из вопроса 16.9 так, чтобы а) использовалась операция прямого чтения, б) пользователь мог вводить номер и количество товара и в) выполнялось вычисление и вывод на экран стоимости (произведение количества товара на стоимость единицы товара).

ГЛАВА 17 Дисковая память III: Расширенные функции DOS

Ц е л ь: Ознакомить с расширенными функциями DOS, начиная с версии 2.0 для обработки дисковых файлов.

ВВЕДЕНИЕ

Функции базовой версии DOS для обработки файлов, показанные в главе 16, действительны для всех последующих версий DOS. В данной главе показаны

ряд расширенных функций, введенных в версиях DOS 2.0 и 3.0 и не поддерживаемых в ранних версиях. Прежде, чем пытаться выполнить дисковые операции из данной главы, следует убедиться в наличии необходимой версии DOS.

Многие из расширенных функций проще своих аналогов в базовой версии DOS. В руководствах по DOS рекомендуется использовать новые функции, которые более естественны для систем типа UNIX. Некоторые операции включают использование строк в формате ASCIIZ для начальной установки дисковода, пути доступа и имени файла; номера файла для последовательного доступа к файлу; специальных кодов возврата.

ДАННЫЕ В ФОРМАТЕ ASCIIZ

При использовании многих расширенных функций для дисковых операций необходимо сообщить DOS адрес строки в формате ASCIIZ, содержащей идентификацию файла в виде номера дисковода, пути доступа и имени файла (все параметры необязательные) и строка должна завершаться шестнадцатеричным нулем, например:

```
PATHNM1   DB   'B:\TEST.ASM',0
PATHNM2   DB   'C:\UTILITY\NU.EXE',0
```

Обратная косая (или прямая косая) используются в качестве разделителя. Нулевой байт (zero) завершает строку (отсюда название ASCIIZ формата). Для прерываний, использующих в качестве параметра ASCIIZ строку, адрес этой строки загружается в регистр DX, например, командой LEA DX,PATHNM1.

ФАЙЛОВЫЙ НОМЕР И КОДЫ ВОЗВРАТА

Операции создания и открытия файла требуют загрузки в регистр AX двухбайтового числа, представляющего собой файловый номер. В главе 8 показано, что стандартные устройства не нуждаются в операции открытия и могут использовать непосредственно файловые номера: 0 - ввод, 1 - вывод, 2 - вывод сообщений об ошибках, 3 - внешнее устройство, 4 - принтер.

Для доступа к диску при создании или открытии файла используется ASCIIZ строка и функции DOS шест.3C или 3D. Успешная операция устанавливает флаг CF в 0 и помещает файловый номер в регистр AX. Этот номер необходимо сохранить в элементе данных DW и использовать его для всех последующих операций над дисковым файлом. При неуспешной операции флаг CF устанавливается в 1, а в регистр AX помещается код ошибки, зависящий от операции (см.табл.17.1).

---

01 Ошибка номера функции
02 Файл не найден
03 Путь доступа не найден
04 Открыто слишком много файлов
05 Нет доступа (Операция отвергнута)
06 Ошибка файлового номера
07 Блок управления памятью разрушен
08 Недостаточно памяти
09 Ошибка адреса блока памяти
10 Ошибка оборудования
11 Ошибка формата
12 Ошибка кода доступа

13 Ошибка данных
15 Ошибка дисковода
16 Попытка удалить оглавление
17 Другое устройство ?
18 Нет больше файлов

Таб.17.1. Коды ошибок

СОЗДАНИЕ ДИСКОВОГО ФАЙЛА

В последующих разделах раскрыты требования к созданию, записи и закрытию дисковых файлов для расширенной версии DOS.

Создание файла: Шест.3С

Для создания нового файла или переписывания старого файла используется функция шест.3С. При этом регистр DX должен содержать адрес ASCIIZ-строки, а регистр CX - необходимый атрибут. Байт атрибут был рассмотрен в главе 15; для обычного файла значение атрибута - 0.

Рассмотрим пример создания обычного файла:

```
MOV  AH,3CH        ;Запрос на создание
MOV  CX,00         ; обычного файла
LEA  DX,PATHNM1    ;ASCIIZ строка
INT  21H           ;Вызов DOS
JC   error         ;Переход по ошибке
MOV  HANDLE1,AX    ;Сохранение файлового номера в DW
```

При правильном открытии операция создает элемент оглавления с данным атрибутом, очищает флаг CF и устанавливает файловый номер в регистре AX. Этот номер должен использоваться для всех последующих операций. Если создаваемый файл уже существует (т.е. имя файла присутствует в оглавлении), то длина этого файла устанавливается в 0 для перезаписи.

В случае возникновения ошибки операция устанавливает флаг CF в 1 и помещает в регистр AX код возврата: 03, 04 или 05 (см.табл.17.1). Код 05 свидетельствует либо о переполнении оглавления, либо о защите существующего файла атрибутом "только чтение". При завершении операции необходимо сначала проверить флаг CF, так как при создании файла возможна установка в регистре AX файлового номера 0005, который можно легко спутать с кодом ошибки 05 (нет доступа).

Запись файла: шест.40

Для записи файла используется функция DOS шест.40. При этом в регистре BX должен быть установлен файловый номер, в регистре CX - число записываемых байт, а в регистре DX - адрес области вывода. В следующем примере происходит запись 256 байт из области OUTREC:

```
HANDLE1 DW   ?
OUTREC  DB   256 DUP (' ')
        MOV  AH,40H        ;Запрос записи
        MOV  BX,HANDLE1    ;Файловый номер
        MOV  CX,256        ;Длина записи
        LEA  DX,OUTREC     ;Адрес области вывода
        INT  21H           ;Вызов DOS
        JC   error2        ;Проверка на ошибку
```

```
               CMP  AX,256        ;Все байты записаны?
               JNE  error3
```

Правильная операция записывает из памяти на диск все данные (256 байт), очищает флаг CF и устанавливает в регистре AX число действительно записанных байтов. Если диск переполнен, то число записанных байтов может отличаться от заданного числа. В случае неправильной операции флаг CF устанавливается в 1, а в регистр AX заносится код 05 (нет доступа) или 06 (ошибка файлового номера).

Закрытие файла: шест.3E

После завершения записи файла необходимо установить файловый номер в регистр BX и, используя функцию DOS шест.3E, закрыть файл. Эта операция записывает все оставшиеся еще данные из буфера на диск и корректирует оглавление и таблицу FAT.

```
          MOV  AH,3EH         ;Запрос на закрытие файла
          MOV  BX,HANDLE1     ;Файловый номер
          INT  21H            ;Вызов DOS
```

В случае ошибки в регистре AX устанавливается код 06 (неправильный файловый номер).

ПРОГРАММА:ИСПОЛЬЗОВАНИЕ ФАЙЛОВОГО НОМЕРА ДЛЯ СОЗДАНИЯ ФАЙЛА.

```
       page    60,132
TTILE   HANCREAT (EXE)   Создание файла на диске
; ---------------------------------------------------
STACKSG SEGMENT PARA STACK 'Stack'
       DW      80 DUP(?)
STACKSG ENDS
; ---------------------------------------------------
DATASG  SEGMENT PARA 'Data'
NAMEPAR LABEL   BYTE                 ;Список параметров:
MAXLEN  DB      30                   ;
NAMELEN DB      ?                    ;
NAMEREC DB      30 DUP('  '), 0DH, 0AH  ; введенное имя, CR/LF для записи
ERRCDE  DB      00
HANDLE  DW      ?
PATHNAM DB      'D:\NAMEFILE.DAT',0
PROMPT  DB      'Name? '
ROW     DB      01
OPNMSG  DB      '*** Open error ***', 0DH, 0AH
WRTMSG  DB      '*** Write error ***', 0DH, 0AH
DATASG  ENDS
; ------------------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
       ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASG
       PUSH    DS
       SUB     AX,AX
       PUSH    AX
       MOV     AX,DATASG
       MOV     DS,AX
       MOV     ES,AX
```

```
       MOV     AX,0600H
       CALL    Q10SCR          ;Очистка экрана
       CALL    Q20CURS         ;Установка курсора
       CALL    C10CREA         ;Создание файла, установка DTA
       CMP     ERRCDE,00       ;Ошибка при создании?
       JZ      A20LOOP         ; да  - продолжить,
       RET                     ; нет - вернуться в DOS
A20LOOP:
       CALL    D10PROC
       CMP     NAMELEN,00      ;Конец ввода?
       JNE     A20LOOP         ; нет - продолжить,
       CALL    G10CLSE         ; да - закрыть файл
       RET                     ;      и выйти в DOS
BEGIN   ENDP
;              Создание файла на диске:
;              -----------------------
C10CREA PROC    NEAR
       MOV     AH,3CH          ;Функция создания файла
       MOV     CX,00           ;Нормальный атрибут
       LEA     CX,PATHNAM
       INT     21H
       JC      C20             ;Есть ошибка?
       MOV     HANDLE,AX       ; нет - запомнить номер,
       RET
C20:                           ; да -
       LEA     DX,OPNMSG       ; выдать сообщение
       CALL    X10ERR          ; об ошибке
       RET
C10CREA ENDP
;              Ввод с клавиатуры:
;              -----------------
D10PROC PROC    NEAR
       MOV     AH,40H          ;Функция вывода на экран
       MOV     BX,01           ;Номер (Handle)
       MOV     CX,06           ;Длина текста запроса
       LEA     DX,PROMPT       ;Выдать запрос
       INT     21H

       MOV     AH,0AH          ;Функция ввода с клавиатуры
       LEA     DX,NAMEPAR      ;Список параметров
       INT     21H
       CMP     NAMELEN,00      ;Имя введено?
       JNE     D20             ; да - обойти
       RET                     ; нет - выйти
D20:
       MOV     AL,20H          ;Пробел для заполнения
       SUB     CH,CH
       MOV     CL,NAMELEN      ;Длина
       LEA     DI,NAMEREC      ;
       ADD     DI,CX           ;Адрес + длина
       NEG     CX              ;Вычислить
       ADD     CX,30           ; оставшуюся длину
       REP     STOSB           ;Заполнить пробелом
D90:
       CALL    F10WRIT         ;Запись на диск
       CALL    E10SCRL         ;Проверка на скроллинг
       RET
D10PROC ENDP
;              Проверка на скроллинг:
;              ---------------------
E10SCRL PROC    NEAR
```

```
       CMP     ROW,18          ;Последняя строка экрана
       JAE     E10             ; да - обойти,
       INC     ROW             ; нет - увеличить строку
       JMP     E10
E10:
       MOV     AX,0601H        ;Продвинуть на одну строку
       CALL    Q10SCR
E90:   CALL    Q20CURS         ;Установка курсора
       RET
E10SCRL ENDP
;              Запись на диск:
;              --------------
F10WRIT PROC    NEAR
       MOV     AH,40H          ;Функция записи на диск
       MOV     BX,HANDLE
       MOV     CX,32           ;30 для имени + 2 для CR/LF
       LEA     DX,NAMEREC
       INT     21H
       JNC     F20             ;Ошибка записи?
       LEA     DX,WRTMSG       ; да -
       CALL    X10ERR          ; выдать предупреждение
       MOV     NAMELEN,00
F20:
       RET
F10WRIT ENDP
;              Закрытие файла на диске:
;              -----------------------
G10CLSE PROC    NEAR
       MOV     NAMEREC,1AH     ;Маркер конца записи (EOF)
       CALL    F10WRIT
       MOV     AH,3EH          ;Функция закрытия
       MOV     BX,HANDLE
       INT     21H
       RET
G10CLSE ENDP
;              Прокрутка (скроллинг) экрана:
;              ----------------------------
Q10SCR  PROC    NEAR            ;в AX - адрес элемента
       MOV     BH,1EH          ;Цвет - желтый на синем
       MOV     CX,0000
       MOV     DX,184FH
       INT     10H             ;Скроллинг
       RET
Q10SCR  ENDP
;              Установка курсора:
;              -----------------
Q20CURS PROC    NEAR
       MOV     AH,02
       MOV     BH,00
       MOV     DH,ROW          ;Установить курсор
       MOV     DL,00
       INT     10H
       RET
Q20CURS ENDP
;              Вывод сообщения об ошибке:
;              -------------------------
X10ERR  PROC    NEAR            ;DX содержит
       MOV     AH,40H          ; адрес сообщения
       MOV     BX,01
       MOV     CX,21           ;Длина сообщения
       INT     21H
```

```
      MOV     ERRCDE,01       ;Установить код ошибки
      RET
X10ERR  ENDP
CODESG  ENDS
      END     BEGIN
```

Рис.17.2. Использование файлового номера для создания файла.

Программа, приведенная на рис.17.2, создает файл по имени, которое вводится пользователем с клавиатуры. В программе имеются следующие основные процедуры:

C10CREA Использует функцию шест.3C для создания файла и сохраняет файловый номер в элементе данных по имени HANDLE.
D10PROC Принимает ввод с клавиатуры и очищает пробелом байты от конца введенного имени до конца области ввода.
F10WRIT Записывает файл, используя функцию шест.40.
G10CLSE В завершении обработки, используя функцию шест.3E,закрывает файл для того, чтобы создать правильный элемент оглавления.

Область ввода имеет длину 30 байтов и завершается двумя байтами: возврат каретки (шест.0DH) и конец строки (шест.0AH). Таким образом общая длина области ввода - 32 байта. Программа переносит на диск 32-х байтовые записи, как записи фиксированной длины. Можно опустить байты "возврат каретки" и "конец строки", но включить их, если потребуется сортировка файла. Программа DOS SORT требует наличия этих байтов для индикации конца записей. Для нашего примера команда SORT может выглядеть следующим образом:

```
SORT   B:<NAMEFILE.DAT   >NAMEFILE.SRT
```

В результате выполнения данной команды записи из файла NAMEFILE.DAT в возрастающей последовательности будут помещены в файл NAMEFILE.SRT. Программа, приведенная на рис.17.3 выполняет чтение записей из файла NAMEFILE.SRT и вывод их на экран. Обратите внимание на два момента: 1) Символы возврат каретки и конец строки включены в конце каждой записи только для выполнения сортировки и в других случаях могут быть опущены. 2) Записи могут иметь переменную длину (по длине вводимых с клавиатуры имен); эта особенность включает некоторое дополнительное программирование, как это будет показано на рис.17.4.

ЧТЕНИЕ ДИСКОВОГО ФАЙЛА

В следующих разделах раскрыты требования для открытия и чтения дисковых файлов в расширенной версии DOS.

Открытие файла: шест.3D

Если в программе требуется прочитать дисковый файл, то прежде необходимо открыть его, используя функцию шест.3D. Эта операция проверяет правильность имени файла и его наличие на диске. При открытии регистр DX должен содержать адрес необходимой ASCIIZ-строки, а регистр AL - код доступа:

0 Открыть файл только для ввода
1 Открыть файл только для вывода

2 Открыть файл для ввода и вывода

Остальные биты регистра AL используются для разделения файлов DOS версии 3.0 и старше (см.техническое руководство по DOS). Обратите внимание, что для записи файла используется функция создания (шест.3C), но не функция открытия файла. Ниже приведен пример открытия файла для чтения:

```
MOV  AH,3DH         ;Запрос на открытие
MOV  AL,00          ;Только чтение
LEA  DX,PATHNM1     ;Строка в формате ASCIIZ
INT  21H            ;Вызов DOS
JC   error4         ;Выход по ошибке
MOV  HANDLE2,AX     ;Сохранение номера в DW
```

Если файл с необходимым именем существует, то операция открытия устанавливает длину записи равной 1, принимает существующий атрибут, сбрасывает флаг CF и заносит файловый номер в регистр AX. Файловый номер используется в дальнейшем для всех последующих операций.

Если файл отсутствует, то операция устанавливает флаг CF и заносит в регистр AX код ошибки: 02, 04, 05 или 12 (см. рис.17.1). Не забывайте проверять флаг CF. При успешном создании файла система может установить в регистре AX файловый номер 0005, что легко можно спутать с кодом ошибки 05 (нет доступа).

Чтение файла: Шест.3F

Для чтения записей файла используется функция DOS шест.3F. При этом необходимо установить в регистре BX файловый номер, в регистре CX - число байтов и в регистре DX - адрес области ввода. В следующем примере происходит считывание 512-байтовой записи:

```
HANDLE2     DW   ?
INPREC      DB   512 DUP (' ')
            MOV  AH,3FH         ;Запрос на чтение
            MOV  BX,HANDLE2     ;Файловый номер
            MOV  CX,512         ;Длина записи
            LEA  DX,INPREC      ;Адрес области ввода
            INT  21H            ;Вызов DOS
            JC   error5         ;Проверка на ошибку
            CMP  AX,00          ;Прочитано 0 байтов?
            JE   endfile
```

Правильно выполненная операция считывает запись в память, сбрасывает флаг CF и устанавливает в регистре AX число действительно прочитанных байтов. Нулевое значение в регистре AX обозначает попытку чтения после конца файла. Ошибочная операция устанавливает флаг CF и возвращает в регистре AX код ошибки 05 (нет доступа) или 06 (ошибка файлового номера).

Так как DOS ограничивает число одновременно открытых файлов, то программа, успешно отработавшая с несколькими файлами, должна закрывать их.

ПРОГРАММА: ИСПОЛЬЗОВАНИЕ ФАЙЛОВОГО НОМЕРА ДЛЯ ЧТЕНИЯ ФАЙЛА

```
      page    60,132
TITLE   HANREAD (EXE)   Чтение записей, созданных в HANCREAT
; ------------------------------------------------
```

```
STACKSG SEGMENT PARA STACK 'Stack'
       DW      80 DUP(?)
STACKSG ENDS
; ----------------------------------------------------
DATASG  SEGMENT PARA 'Data'
ENDCDE  DB      00
HANDLE  DW      ?
IOAREA  DB      32 DUP(' ')
PATHNAM DB      'D:\NAMEFILE.SRT',0
OPENMSG DB      '*** Open error ***', 0DH, 0AH
READMSG DB      '*** Read error ***', 0DH, 0AH
ROW     DB      00
DATASG  ENDS
; ----------------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
       ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASG
       PUSH    DS
       SUB     AX,AX
       PUSH    AX
       MOV     AX,DATASG
       MOV     DS,AX
       MOV     ES,AX
       MOV     AX,0600H
       CALL    Q10SCR          ;Очистить экран
       CALL    Q20CURS         ;Установить курсор
       CALL    E100PEN         ;Открыть файл,
;                              ; установить DTA
       CMP     ENDCDE,00       ;Ошибка открытия?
       JNZ     A90             ; да - завершить программу
A20LOOP:
       CALL    F10READ         ;Чтение записи с диска
       CMP     ENDCDE,00       ;Ошибка чтения?
       JNZ     A90             ; да - выйти,
       CALL    G10DISP         ; нет - выдать имя,
       JMP     A20LOOP         ;       и продолжить
A90:   RET
BEGIN   ENDP
;              Открытие файла:
;              --------------
E100PEN PROC    NEAR
       MOV     AH,3DH          ;Функция открытия
       MOV     CX,00           ;Нормальные атрибуты
       LEA     DX,PATHNAM
       INT     21H
       JC      E20             ;Ошибка открытия?
       MOV     HANDLE,AX       ; нет - сохранить
       RET                     ;  файловый номер
E20:
       MOV     ENDCDE,01       ; да - выдать
       LEA     DX,OPENMSG      ;  сообщение об ошибке
       CALL    X10ERR
       RET
E100PEN ENDP
;              Чтение дисковой записи:
;              ----------------------
F10READ PROC    NEAR
       MOV     AX,3FH          ;Функция чтения
       MOV     BX,HANDLE
       MOV     CX,32           ;30 для имени, 2 для CR/LF
       LEA     DX,IOAREA
```

```
        INT     21H
        JC      F20             ;Ошибка при чтении?
        CMP     AX,00           ;Конец файла?
        JE      F30
        CMP     IOAREA,1AH      ;Маркер конца файла (EOF)?
        JE      F30             ; да - выйти
        RET
F20:
        LEA     DX,READMSG      ; нет - выдать
        CALL    X10ERR          ;  сообщение об ошибке
F30:
        MOV     ENDCDE,01       ;Код завершения
F90:    RET
F10READ ENDP
;            Вывод имени на экран:
;            --------------------
G10DISP PROC    NEAR
        MOV     AH,40H          ;Функция вывода на экран
        MOV     BX,01           ;Установить номер
        MOV     CX,32           ; и длину
        LEA     DX,IOAREA
        INT     21H
        CMP     ROW,20          ;Последняя строка экрана?
        JEA     G90             ; да - обойти
        INC     ROW
        RET
G90:
        MOV     AX,0601H
        CALL    Q10SCR          ;Прокрутка (скроллинг)
        CALL    Q20CURS         ;Установить курсор
        RET
G10DISP ENDP
;            Прокрутка (скроллинг) экрана:
;            ----------------------------
Q10SCR  PROC    NEAR            ;в AX - адрес элемента
        MOV     BH,1EH          ;Установить цвет
        MOV     CX,0000
        MOV     DXX,184FH       ;Функция прокрутки
        INT     10H
        RET
Q10SCR  ENDP
;            Установка курсора:
;            -----------------
Q20CURS PROC    NEAR
        MOV     AH,02           ;Функция установки курсора
        MOV     BH,00           ; курсор
        MOV     DH,ROW          ; строка
        MOV     DL,00           ; столбец
        INT     10H
        RET
Q20CURS ENDP
;            Вывод сообщения об ошибке:
;            ------------------------
X10ERR  PROC    NEAR
        MOV     AH,40H          ;в DX - адрес сообщения
        MOV     BX,01           ;Номер
        MOV     CX,20           ;Длина сообщения
        INT     21H
        RET
X10ERR  ENDP
```

```
CODESG  ENDS
        END     BEGIN
```

Рис.17.3. Использование файлового номера для чтения файла.

На рис.17.3 приведена программа, которая читает файл, созданный предыдущей программой (см.рис.17.2) и отсортированный командой DOS SORT. Для открытия файла используется функция шест.3D. Полученный в результате файловый номер заносится в поле HANDLE и используется затем в функции шест.3F для чтения файла.

В программе нет необходимости переносить курсор на новую строку, так как записи содержат в конце символы "возврат каретки" и "новая строка".

ASCII-ФАЙЛЫ (ФАЙЛЫ В ФОРМАТЕ ASCII)

В предыдущих примерах были показаны операции создания и чтения файлов. Аналогичным образом можно обрабатывать ASCII-файлы (текстовые файлы), созданные DOS или редактором. Для этого необходимо знать организацию оглавления и таблицы FAT, а также способ записи данных в сектор диска, используемый системой. Система DOS записывает, например, ASM-файл в точном соответствии с вводом с клавиатуры, включая символы табуляции (шест.09), возврат каретки (шест.OD) и конец строки (шест.OA). Для экономии дисковой памяти DOS не записывает пробелы, которые находятся на экране и предшествуют символу табуляции, и пробелы, находящиеся в строке справа от символа "возврат каретки". Следующий пример иллюстрирует ассемблерную команду, как она может выглядеть на экране:

```
<tab>MOV<tab>AH,09<return>
```

Для такой строки содержимое ASCII-файла будет:

```
094D4F560941482C30390D0A
```

Когда программа TYPE или редактор читают файл и выводят на экран символы "табуляция", "возврат каретки" и "конец строки" автоматически выравнивают данные.

Рассмотрим программу, приведенную на рис.17.4, которая читает и выводит на экран файл HANREAD.ASM (пример на рис. 17.3) по секторам. Если программа HANREAD уже введена и проверена, то можно просто скопировать ее в файл с новым именем.

```
        page    60,132
TITLE   ASCREAD (COM)   Чтение ASCII файла
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:CODESG
        ORG     100H
BEGIN:  JMP     MAIN
; -----------------------------------------------
SECTOR  DB      512 DUP(' ')    ;Область ввода
DISAREA DB      120 DUP(' ')    ;Область вывода на экран
ENDCDE  DW      00
HANDLE  DW      0
OPENMSG DB      '*** Open error ***'
PATHNAM DB      'D:\HANREAD.ASM', 0
ROW     DB      00
```

```
; -------------------------------------------------
MAIN    PROC    NEAR            ;Основная программа
        MOV     AX,0600H
        CALL    Q10SCR          ;Очистить экран
        CALL    Q20CURS         ;Установить курсор
        CALL    E10OPEN         ;Открыть файл,
                                ; установить DTA
        CMP     ENDCDE,00       ;Ошибка при открытии?
        JNE     A90             ; да - выйти,
A20LOOP:                        ; нет - продолжить
        CALL    R10READ         ;Чтение первого сектора
        CMP     ENDCDE,00       ;Конец файла, нет данных?
        JE      A90             ; да - выйти
        CALL    G10XPER         ;Выдать на экран
A90:    RET                     ;Завершить программу
MAIN    ENDP
;               Открыть файл на диске:
;               ---------------------
E10OPEN PROC    NEAR
        MOV     AH,3DH          ;Функция открытия
        MOV     AL,00           ;Только чтение
        LEA     DX,PATHNAM
        INT     21H
        JNC     E20             ;Проверить флаг CF
        CALL    X10ERR          ; ошибка, если установлен
        RET
E20:
        MOV     HANDLE,AX       ;Запомнить номер файла
        RET
E10OPEN ENDP
;               Построчный вывод данных на экран:
;               --------------------------------
G10XPER PROC    NEAR
        CLD                     ;Направление слева-направо
        LEA     SI,SECTOR
G20:
        LEA     DI,DISAREA
G30:
        LEA     DX,SECTOR+512
        CMP     SI,DX           ;Конец сектора?
        JNE     G40             ; нет - обойти,
        CALL    R10READ         ; да - читать следующий
        CMP     ENDCDE,00       ;Конец файла?
        JE      G80             ; да - выйти
        LEA     SI,SECTOR
G40:
        LEA     DX,DISAREA+80
        CMP     DI,DX           ;Конец DISAREA?
        JB      G50             ; нет - обойти,
        MOV     [DI],0D0AH      ; да - установить CR/LF
        CALL    H10DISP         ;  и выдать на экран
        LEA     DI,DISAREA
G50:
        LODSB                   ;Загрузить [SI] в AL
                                ; и увеличить SI
        STOSB                   ;Записать AL в [DI]
                                ; и увеличить DI
        CMP     AL,1AH          ;Конец файла?
        JE      G80             ; да - выйти
        CMP     AL,0AH          ;Конец строки?
        JNE     G30             ; нет - повторить цикл,
```

```
        CALL    H10DISP         ; да - вывести на экран
        JMP     G20
G80:
        CALL    H10DISP         ;Вывести последнюю строку
G90:    RET
G10XPER ENDP
;               Вывод строки на экран:
;               --------------------
H10DISP PROC    NEAR
        MOV     AH,40H          ;Функция вывода на экран
        MOV     BX,01           ;Номер (Handle)
        LEA     CX,DISAREA      ;Вычислить
        NEG     CX              ; длину
        ADD     CX,DI           ; строки
        LEA     DX,DISAREA
        INT     21H
        CMP     ROW,22          ;Последняя строка экрана?
        JAE     H20             ; нет - выйти
        INC     ROW
        JMP     H90
H20:
        MOV     AX,0601H        ;Прокрутка (скроллинг)
        CALL    Q10SCR
        CALL    Q20CURS
H90:    RET
H10DISP ENDP
;               Чтение дискового сектора:
;               ------------------------
R10READ PROC    NEAR
        MOV     AH,3FH          ;Функция чтения
        MOV     BX,HANDLE       ;Устройство
        MOV     CX,512          ;Длина
        LEA     DX,SECTOR       ;Буфер
        INT     21H
        MOV     ENDCDE,AX
        RET
R10READ ENDP
;               Прокрутка (скроллинг) экрана:
;               ----------------------------
Q10SCR  PROC    NEAR            ;в AX адрес элемента
        MOV     BH,1EH          ;Установить цвет
        MOV     CX,0000         ;Прокрутка
        MOV     DX,184FH
        INT     10H
        RET
Q10SCR  ENDP
;               Установка курсора:
;               -----------------
Q20CURS PROC    NEAR
        MOV     AH,02           ;Функция
        MOV     BH,00           ; установки курсора
        MOV     DH,ROW
        MOV     DL,00
        INT     10H
        RET
Q20CURS ENDP
;               Вывод сообщения об ошибке на диске:
;               ---------------------------------
X10ERR  PROC    NEAR
        MOV     AH,40H          ;Функция вывода на экран
        MOV     BX,01           ;Номер устройства
```

```
       MOV     CX,18           ;Длина
       LEA     DX,OPENMSG
       INT     21H
       MOV     ENDCDE,O1       ;Индикатор ошибки
       RET
X10ERR  ENDP

CODESG  ENDS
       END     BEGIN
```

Рис.17.4. Чтение ASCII-файла.

Программа выполняет в основном те же функции, что и DOS TYPE, т.е. выводит на экран каждую запись до символов "возврат каретки" и "конец строки" (CR/LE). Прокрутка содержимого экрана (скроллинг) вызывает некоторые проблемы. Если в программе не будет предусмотрено специальной проверки на конец экрана, то вывод новых строк будет осуществляться поверх старых и при короткой длине старые символы будут оставаться справа от новой строки. Для правильной прокрутки необходимо подсчитывать строки и контролировать достижение конца экрана. Так как строки ASCII-файла имеют переменную длину, то следует определять конец каждой строки прежде, чем выводить ее на экран.

Рассматриваемая программа считывает полный сектор данных в область SECTOR. Процедура G10XFER передает данные побайтно из области SECTOR в область DISAREA, откуда они будут выдаваться на экран. При обнаружении символа "конец строки", процедура выводит на экран содержимое DISAREA, включая "конец строки". (Экран дисплея принимает также символы табуляции (шест.09) и автоматически устанавливает курсор в следующую справа позицию кратную 8).

В программе необходимо проверять конец сектора (для считывания следующего) и конец области вывода. Для стандартных ASCII-файлов, таких как ASM-файлы, каждая строка имеет относительно короткую длину и гарантировано завершается парой символов CR/LF. Нетекстовые файлы, такие как EXE или OBJ, не имеют строк и поэтому рассматриваемая программа должна проверять достижение конца области DISAREA во избежание разрушения. Хотя программа предназначена для вывода на экран только ASCII-файлов, она имеет проверку для страховки от всяких неожиданных несимвольных файлов.

Процедура G10XFER выполняет следующее:

1. Инициализирует адрес области SECTOR.
2. Инициализирует адрес области DISAREA.
3. При достижении конца области SECTOR считывает следующий сектор. В случае конца файла, завершает работу программы, иначе инициализирует адрес области SECTOR.
4. При достижении конца области DISAREA вставляет символы CR/LF, выводит строку на экран и инициализирует адрес DISAREA.
5. Переписывает символ из области SECTOR в область DISAREA.
6. По символу "конец файла" (шест.1A) завершает работу программы.
7. По символу "конец строки" (шест.OA) выводит на экран строку и переходит на п.2, по другим символам идет на п.3.

Попробуйте выполнить эту программу в отладчике DEBUG. При каждом вводе с диска просмотрите содержимое области ввода и обратите внимание на то, как DOS форматирует записи. Для улучшения данной программы организуйте вывод на экран запроса для указания пользователем имени и типа файла.

ДРУГИЕ ДИСКОВЫЕ ФУНКЦИИ В РАСШИРЕННОЙ ВЕРСИИ DOS

Получение размера свободного дискового пространства: шест.36

Данная функция выдает информацию о дисковой памяти. Для выполнения функции необходимо загрузить в регистр DL номер дисковода (0 - текущий дисковод, 1 - A, 2 - B и т.д.):

```
MOV  AH,36H    ;Запрос на
MOV  DL,0      ; текущий дисковод
INT  21H       ;Вызов DOS
```

При указании неправильного номера дисковода операция возвращает в регистре AX шест.FFFF, иначе следующие значения:

в AX число секторов на кластер,
в BX число доступных кластеров,
в CX число байтов на сектор,
в DX общее число кластеров на дисководе.

В версии DOS младше 2.0 для получения информации о дисковой памяти следует использовать функцию шест.1B (получить информацию из таблицы FAT).

Удаление файла: шест.41

Для удаления файлов из программы (за исключением файлов с атрибутом "только чтение") используется функция шест.41. При этом в регистре DX необходимо загрузить ASCIIZ строку, содержащую путь доступа и имя файла:

```
MOV  AH,41H        ;Запрос на удаление
LEA  DX,PATHNAM    ;ASCIIZ-строка
INT  21H           ;Вызов DOS
```

В случае ошибки в регистре AX возвращается код 02 (файл не найден) или 05 (нет доступа).

Управление файловым указателем: шест.42

Система DOS имеет файловый указатель, который при открытии файла устанавливается в 0 и увеличивается на 1 при последовательных операциях записи или считывания. Для доступа к любым записям внутри файла можно менять файловый указатель с помощью функции шест.42, получая в результате прямой доступ к записям файла.

Для установки файлового указателя необходимо поместить в регистр BX файловый номер и в регистровую пару CX:DX требуемое смещение в байтах. Для смещений до 65.535 в регистре CX устанавливается 0, а в DX - смещение. В регистре AL должен быть установлен один из кодов, который определяет точку отсчета смещения:

0 - смещение от начала файла.
1 - смещение текущего значения файлового указателя, которое может быть в любом месте, включая начало файла.
2 - смещение от конца файла. Размер файла (и следовательно смещение до конца файла) можно определить, установив регистровую пару CX:DX в 0 и используя код 2 в регистре AL.

В следующем примере устанавливается файловый указатель на смещение 1024 байта от начала файла:

```
MOV  AH,42H        ;Установка указателя
```

```
        MOV  AL,00         ; от начала файла
        LEA  BX,HANDLE1     ;Установка файлового номера
        MOV  CX,00         ;
        MOV  DX,1024       ;Смещение 1024 байта
        INT  21H           ;Вызов DOS
        JC   error
```

Правильно выполненная операция сбрасывает флаг CF и возвращает новый указатель в регистровой паре DX:AX. Неправильная операция устанавливает флаг CF в 1 и возвращает в регистре AX код 01 (ошибка кода отсчета) или 06 (ошибка файлового номера).

Проверка или изменение атрибута: шест.43

Для проверки или изменения файлового атрибута в оглавлении диска используется функция шест.43H. При этом в регистре DX должен быть установлен адрес ASCIIZ строки. Для проверки атрибута регистр AL должен содержать 00. Для изменения атрибута регистр AL должен содержать 01, а регистр CX - новое значение атрибута. Следующий пример устанавливает нормальный атрибут:

```
        MOV  AH,43H        ;Запрос на установку
        MOV  AL,01         ; нормального
        MOV  CX,00         ; атрибута
        LEA  DX,PATHNM2     ;ASCIIZ-строка
        INT  21H           ;Вызов DOS
```

В случае проверки функция возвращает текущий атрибут файла в регистре CX. В случае изменения функция устанавливает в соответствующем элементе оглавления атрибут из регистра CX. Неправильная операция возвращает в регистре AX коды ошибок 02, 03 или 05.

Получить текущее оглавление: шест.47

Определение текущего оглавления для любого дисковода осуществляется с помощью функции шест.47. При этом необходимо определить область памяти достаточно большую, чтобы содержать пути доступа максимальной длины и загрузить адрес этой области в регистр DX. Регистр DL должен содержать номер дисковода: 0 - текущий, 1 - A, 2 - B и т.д. В результате выполнения операция помещает в область памяти имя текущей директории (без номера дисковода), например, в следующем виде:

```
        ASSEMBLE\EXAMPLES
```

Нулевой байт (шест.00) идентифицирует конец составного имени пути доступа. Для корневой директории возвращаемое значение состоит только из одного байта - шест.00. Таким образом можно получить текущее имя пути доступа для любого файла в подоглавлении. Пример на рис.17.5 демонстрирует использование данной функции.

```
TITLE   GETPATH (COM)   Получить текущий каталог
CODESG  SEGMENT PARA 'Code'
      ASSUME  CS:CODESG,DS:CODESG,ES:CODESG
      ORG     100H
BEGIN:  JMP     SHORT MAIN
; ---------------------------------------------
PATHNAM DB      65 DUP(' ')     ;Имя текущего пути доступа
```

```
; ------------------------------------------
MAIN    PROC    NEAR
        MOV     AH,19H          ;Определить текущий диск
        INT     21H
        ADD     AL,41H          ;Заменить шест.номер
        MOV     DL,AL           ; на букву: 0=A, 1=B ...
        CAL     B10DISP         ;Выдать номер дисковода,
        MOV     DL,':'
        CAL     B10DISP         ; двоеточие,
        MOV     DL,'\'
        CAL     B10DISP         ; обратную косую

        MOV     AH,47H          ;Получить текущий каталог
        MOV     DL,00
        LEA     SI,PATHNAM
        INT     21H
A10LOOP:
        CMP     BYTE PTR [SI],0 ;Конец имени пути доступа
        JE      A20             ; да - выйти
        MOV     AL,[SI]         ;Выдать на экран
        MOV     DL,AL           ; имя пути доступа
        CALL    B10DISP         ; побайтно
        INC     SI
        JMP     A10LOOP
A20:    RET                     ;Выход в DOS
MAIN    ENDP

B10DISP PROC    NEAR
        MOV     AH,02           ;в DL - адрес элемента
        INT     21H             ;Функция вывода на экран
        RET
B10DISP ENDP

CODESG  ENDS
        END     BEGIN
```

Рис.17.5. Получить текущий каталог.

Поиск файлов по шаблону: шест.4E и шест.4F

Данные функции аналогичны функциям шест. 11 и 12 базовой версии DOS. Функция 4E используется для начала поиска в оглавлении, а функция 4F - для продолжения. Для начала поиска в регистр DX необходимо загрузить адрес ASCIIZ-строки, содержащей имя пути доступа и шаблон поиска. Шаблон поиска может включать в себя символы ? и *. В регистре CX должно быть значение атрибута в любой комбинации битов (нормальный, оглавление, спрятанный или системный).

```
MOV  AH,4EH        ;Запрос на начало поиска
MOV  CX,00H        ;Нормальный атрибут
LEA  DX,PATHNM1    ;ASCIIZ-строка
INT  21H           ;Вызов DOS
```

Если операция находит файл, удовлетворяющий шаблону поиска, то в текущий буфер DTA в FCB заполняется следующей информацией:

00 - резервировано DOS для последующего поиска
21 - атрибут файла
22 - время файла

24 - дата файла
26 - размер файла: младшее слово, затем старшее слово
30 - имя и тип в виде 13-байтовой ASCIIZ строки, завершаемой шест.00.

В случае ошибки в регистре AX возвращается код 02 (не найдено) или 18 (нет больше файлов). Для продолжения поиска файлов (после функции шест.4E) используется функция 4F. Между этими функциями не следует нарушать содержимое DTA.

```
        MOV  AH,4FH         ;Запрос на продолжение поиска
        INT  21H           ;Вызов DOS
```

Единственно возможный код в регистре AX - 18 (нет больше файлов). Обе рассмотренные функции не меняют состояние флага CF.

Переименование файла: шест.56

Для переименования файла используется функция шест.56. При этом в регистр DX должен быть загружен адрес ASCIIZ-строки, содержащей старые значения дисковода, пути доступа, имени и типа файла, а в регистр DI (в действительности ES:DI) - адрес ASCIIZ-строки, содержащей новые значения дисковода, пути доступа, имени и типа файла. Если указывается номер дисковода, то он должен быть одинаков в обоих строках. Путь доступа может быть различным, поэтому данная операция может не только переименовывать файл, но и переносить его в другое подоглавление.

```
        MOV  AH,56H         ;Запрос на переименование файла
        LEA  DX,oldstring   ;DS:DX
        LEA  DI,newstring   ;ES:DI
        INT  21H           ;Вызов DOS
```

В случае ошибки регистр AX возвращает коды 03 (путь доступа не найден), 05 (нет доступа?) и 17 (разные дисководы).

Другие функции DOS, имеющие отношение к дисковым файлам, включают создание подоглавления (шест.39), удаление элемента оглавления (шест.3A), изменение текущего оглавления (шест. 3B), управление вводом-выводом для устройств (шест.44), дублирование файлового номера (шест.45), принудительное дублирование файлового номера (шест.46), получение состояния проверки ? (шест.54).

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Многие функции расширенной версии DOS оперируют с ASCIIZ-строками, которые содержат путь доступа и завершаются байтом, содержащим шест.00.

- Функции создания и открытия возвращают значение файлового номера, который используется для последующего доступа к файлу.

- В случае ошибок многие функции устанавливают флаг CF и помещают код ошибки в регистр AX.

- Как правило, функция создания используется для записи файла, а открытия - для чтения.

- После того, как файл записан на диск, его необходимо закрыть для того, чтобы в оглавление были внесены соответствующие изменения.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

17.1. Какие значения кодов возврата для ситуаций "файл не найден" и "ошибка файлового номера" ?

17.2. Определите ASCIIZ-строку по имени PATH1 для файла CUST.LST на дисководе C.

17.3. Для предыдущего файла (п.17.2) напишите команды а) определения элемента по имени CUSTHAN для файлового номера, б) создание файла, в) записи файла из области CUSTOUT (128 байт) и г) закрытия файла. Обеспечьте проверку на ошибки.

17.4. Для файла (п.17.3) напишите команды а) открытия файла и б) чтения файла в область CUSTIN. Обеспечьте контроль ошибок.

17.5. В каких случаях необходимо закрывать файл, который был открыт только для чтения ?

17.6. Измените программу на рис.17.4 так, чтобы пользователь мог вводить с клавиатуры имя файла, который необходимо выдать на экран. Обеспечьте возможность любого числа запросов и завершение программы только по пустому запросу, т.е. простому нажатию клавиши Return.

ГЛАВА 18 Дисковая память IV: Функции BIOS

Ц е л ь: Показать основные требования к программированию функций BIOS для создания и чтения дисковых файлов.

ВВЕДЕНИЕ

Для дисковых операций можно программировать непосредственно на уровне BIOS, хотя BIOS и не обеспечивает автоматически использование оглавления или блокирование/деблокирование записей. Дисковая операция BIOS INT 13H рассматривает все "записи", как имеющие размер сектора, а адресацию диска осуществляет в терминах действительных номера дорожки и номера сектора.

Для дисковых операций чтения, записи и верификации необходима инициализация следующих регистров:

AH Определяет тип операции: чтение, запись, верификация или форматирование.
AL Определяет число секторов.
CH Определяет номер дорожки.
CL Определяет номер начального сектора.
DH Номер головки (стороны) : 0 или 1 для дискеты.
DL Номер дисковода: 0=A, 1=B и т.д.
ES:BX Адрес буфера ввода/вывода в области данных (за исключением операции верификации).

ДИСКОВЫЕ ОПЕРАЦИИ В BIOS

Для указания необходимой дисковой операции необходимо перед INT 13H

загрузить в регистр AH соответствующий код.

AH = 00: Сброс системы контролера дисковода

Данная операция осуществляет полный сброс контролера дисковода и требует для выполнения INT 13H загрузку в регистр AH значение шест.00. Операция используется в случаях, когда после других дисковых операций возвращается код серьезной ошибки.

AH = 01: Определить состояние дисковода

Данная операция возвращает в регистре AL состояние дисковода после последней операции ввода/вывода (см.Байт состояния в следующем разделе). Операция требует только загрузки значения 01 в регистр AH.

AH = 02: Чтение секторов

Данная операция выполняет чтение в память определенного числа секторов на одной дорожке. Число секторов обычно 1, 8 или 9. Адрес памяти для области ввода должен быть загружен в регистр BX, причем следует помнить, что реальный адрес зависит от содержимого регистра EX, так как в данном случае используется регистровая пара ES:BX. В следующем примере выполняется чтение сектора в область INSECT, которая должна быть достаточно большой, чтобы вместить все данные:

```
MOV  AH,02     ;Запрос на чтение
MOV  AL,01     ; один сектор
LEA  BX,INSERT ;Буфер ввода в ES:BX
MOV  CH,05     ;Дорожка 05
MOV  CL,03     ;Сектор 03
MOV  DH,00     ;Сторона (головка) 00
MOV  DL,01     ;Дисковод 01 (B)
INT  13H       ;Вызов BIOS
```

Число действительно прочитанных секторов возвращается в регистре AL. Регистры DS, BX, CX и DX сохраняют свои значения.

В большинстве случаев программа указывает только один сектор или все сектора на дорожке. Для последовательного чтения секторов программа должна увеличивать содержимое регистров CH и CL. Заметьте, что когда номер сектора достигает максимального значения, его необходимо сбросить в 01, а номер дорожки увеличить на 1 или изменить сторону 0 на 1 (для двухсторонних дискет).

AH = 03: Запись секторов

Данная операция записывает данные из указанной области памяти (обычно 512 байтов или кратное 512) в один или несколько определенных секторов. Управляющая информация загружается в регистры аналогично операции чтения диска (код 02). Операция записи возвращает в регистре AL число секторов, которые действительно были записаны. Регистры DX, BX, CX и DX сохраняют свои значения.

AH = 04: Верификация сектора

Данная операция проверяет, может ли быть найден указанный сектор, и выполняет своего рода контроль на четность. Операцию можно использовать

после записи (код 03) для гарантии более надежного вывода, на что потребуется дополнительное время ввода/вывода. Значения регистров устанавливаются аналогично операции записи (код 03), за исключением регистровой пары ES:BX - их инициализация не требуется. Операция возвращает в регистре AL число обработанных секторов. Регистры DX, BX, CX и DX сохраняют свои значения.

AH = 05: Форматирование дорожек

Данная операция используется для форматирования определенного числа дорожек в соответствии с одним из четырех размеров (стандарт для системы PC - 512). Операции чтения и записи для локализации требуемого сектора требуют информацию о формате. Для форматирования регистровая пара ES:BX должна содержать адрес, который указывает на группу адресных полей для дорожки. Для каждого сектора на дорожке должен быть четырехбайтовый элемент в виде T/H/S|B, где

T номер дорожки,
H номер головки,
S номер сектора,
B число байт на секторе, (00-128, 01-256, 02-512, 03-1024).

Например, для форматирования 03 дорожки, на стороне 00 и 512 байтов на сектор, первый элемент должен иметь значение шест.03000102 и за ним должны быть описаны элементы для остальных секторов на дорожке. Техническое руководство по AT содержит ряд дополнительных операций BIOS.

БАЙТ СОСТОЯНИЯ

Для всех рассмотренных выше операций (02, 03, 04 и 05) в случае нормального завершения флаг CF и регистр AH содержит 0. В случае ошибки флаг CF устанавливается в 1, а регистр AH содержит код состояния, идентифицирующий причину ошибки. Код состояния аналогичен значению в регистре AL после выполнения операции 01.

AH	Причина
0000 0001	Ошибка команды для дискеты
0000 0010	Не найден адресный маркер на диске
0000 0011	Попытка записи на защищенный диск
0000 0100	Не найден сектор
0000 1000	Выход за границы DMA (памяти прямого доступа)
0000 1001	Попытка доступа через границу 64K
0001 0000	Чтение сбойный участок на диске
0010 0000	Ошибка контролера дисковода
0100 0000	Ошибка установки (поиска)
1000 0000	Ошибка оборудования

В случае возникновения ошибки, обычным действием является сброс диска (AH=00) и троекратное повторение операции. Если таким образом ошибка не устраняется, то на экран выводится соответствующее сообщение и пользователь может сменить дискету.

ПРОГРАММА: ИСПОЛЬЗОВАНИЕ BIOS ДЛЯ ЧТЕНИЯ СЕКТОРОВ

```
TITLE   BIOREAD (COM)   Чтение дискового сектора через BIOS
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:CODESG
        ORG     100H
BEGIN   JMP     MAIN
; --------------------------------------------------------
RECDIN  DB      512 DUP(' ')    ;Область ввода
ENDCDE  DB      00
CURADR  DW      0304H           ;Начало (дорожка/сектор)
ENDADR  DW      0501H           ;Конец (дорожка/сектор)
READMSG DB      '*** Read error ***$'
SIDE    DB      00
; --------------------------------------------------------
MAIN    PROC    NEAR
        MOV     AX,0600H        ;Функция прокрутки экрана
A20LOOP:
        CALL    Q10SCR          ;Очистить экран
        CALL    Q20CURS         ;Установить курсор
        CALL    C10ADDR         ;Определить адрес на диске
        MOV     CX,CURADR
        MOV     DX,ENDADR
        CMP     CX,DX           ;Последний сектор?
        JE      A90             ; да - выйти
        CALL    F10READ         ;Получить дисковую запись
        CMP     ENDCDE,00       ;Ошибка чтения?
        JNZ     A90             ; да - выйти
        CALL    G10DISP         ;Вывести сектор на экран
        JMP     A20LOOP         ;Повторить
A90     RET                     ;Завершить программу
MAIN    ENDP
;               Вычислить следующий адрес на диске:
;               ----------------------------------
C10ADDR PROC    NEAR
        MOV     CX,CURADR       ;Последняя дорожка/сектор
        CMP     CL,10           ;Последний сектор?
        JNE     C90             ; нет - выйти
        CMP     SIDE,00         ;Обойти, если сторона = 0
        JE      C20
        INC     CH              ;Увеличить номер дорожки
C20:
        XOR     SIDE,01         ;Сменить сторону
        MOV     CL,01           ;Установить сектор = 1
        MOV     CURADR,CX
C90:    RET
C10ADDR ENDP
;               Чтение дискового сектора:
;               ------------------------
F10READ PROC    NEAR
        MOV     AL,01           ;Число секторов
        MOV     AH,02           ;Функция чтения
        LEA     BX,RECDIN       ;Адрес буфера
        MOV     CX,CURADR       ;Дорожка/сектор
        MOV     DH,SIDE         ;Сторона
        MOV     DL,01           ;Дисковод B
        INT     13H             ;Выполнить ввод
        CMP     AH,00           ;Ошибка чтения?
        JZ      F90             ; нет - выйти
        MOV     ENDCDE,01       ; да -
        CALL    X10ERR          ;  ошибка чтения
F90:
```

```
        INC     CURADR          ;Увеличить номер сектора
        RET
F10READ ENDP
;               Вывод сектора на экран:
;               ---------------------
G10DISP PROC    NEAR
        MOV     AH,40H          ;Функция вывода на экран
        MOV     BX,01           ;Номер устройства
        MOV     CX,512          ;Длина
        LEA     DX,RECDIN
        INT     21H
        RET
G10DISP ENDP
;               Очистка экрана:
;               --------------
Q10SCR  PROC    NEAR
        MOV     AX,0600H        ;Полный экран
        MOV     BH,1EH          ;Установить цвет
        MOV     CX,0000         ;Функция прокрутки
        MOV     DX,184FH
        INT     10H
        RET
Q10SCR  ENDP
;               Установка курсора:
;               -----------------
Q20CURS PROC    NEAR
        MOV     AH,02           ;Функция установки
        MOV     BH,00           ; курсора
        MOV     DX,0000
        INT     10H
        RET
Q20CURS ENDP
;               Вывод сообщения об ошибке на диске:
;               ---------------------------------
X10ERR  PROC    NEAR
        MOV     AH,40H          ;Функция вывода на экран
        MOV     BH,01           ;Номер устройства
        MOV     CX,18           ;Длина сообщения
        LEA     DX,READMSG
        INT     21H
        RET
X10ERR  ENDP
CODESG  ENDS
        END     BEGIN
```

Рис.18.1. Использование BIOS для чтения дискового файла.

Рассмотрим программу, приведенную на рис.18.1, в которой используется команда BIOS INT 13H для чтения секторов диска. Программа базируется на примере, приведенном на рис.16.3, со следующими изменениями:

1. Отсутствует описание FCB и подпрограмма открытия.
2. Программа расчитывает каждый дисковый адрес. После каждого чтения происходит увеличение номера сектора. При достижении номера сектора 10 процедура C10ADDR сбрасывает это значение в 01. Если номер стороны = 1, программа увеличивает номер дорожки; затем меняется номер стороны: 0 на 1 и 1 на 0.
3. Область CURADR содержит начальные значения номеров дорожки и сектора (их программа увеличивает), а область ENDADR - конечные

значения. Один из способов улучшения программы - предоставить пользователю возможность указать начальные и конечные номера дорожки и сектора с помощью соответствующего запроса.

Выполните данную программу под управлением отладчика DEBUG. Проделайте трассировку команд, которые инициализируют сегментные регистры, и установите начальный и конечный номера секторов для файловой таблицы FAT (расположение таблицы FAT различно в разных версиях операционной системы). Используя команду G (до) для выполнения ввода с диска и проверки считанного содержимого таблицы FAT и элементов оглавления.

В качестве альтернативы отладчику DEBUG можно преобразовать ASCII-символы в области ввода в их шест. эквиваленты и выдать на экран эти значения, как это делает отладчик DEBUG (см. программу на рис.14.5). Таким образом можно проверить содержимое любого сектора (в том числе "спрятанного"), а также предоставить пользователю возможность внести изменения и записать измененный сектор на диск.

Следует помнить, что при создании файла DOS может вносить записи на любые доступные сектора, которые не обязательно будут смежными на диске. Следовательно, с помощью команды BIOS INT 13H нельзя выполнить последовательное чтение файла.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Команда BIOS INT 13H обеспечивает прямой доступ к дорожкам и секторам диска.

- Команда BIOS INT 13H не поддерживает операции с оглавлением, обнаружение конца файла, блокирование и деблокирование записей.

- Верификация сектора выполняет элементарную проверку записанных данных, что приводит к увеличению времени обработки.

- Проверяйте байт состояния после каждой дисковой операции через BIOS.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

18.1. Напишите команды для сброса дискового контролера.

18.2. Напишите команды для чтения байта состояния дискеты.

18.3. Напишите команды для BIOS INT 13H, выполняющие чтение одного сектора в область памяти INDISK, с дисковода A, головки 0, дорожки 6 и сектора 3.

18.4. Напишите команды для BIOS INT 13H, выполняющие запись трех секторов из области памяти OUTDISK, на дисковод B, головку 0, дорожку 8 и сектор 1.

18.5. При записи данных в вопросе 18.4, как можно распознать попытку записи на защищенный диск?

18.6. На основе вопроса 18.4 напишите команды контроля записи (операция верификации).

Ц е л ь: Описать возможности программ на языке ассемблера для вывода информации на печатающее устройство (принтер).

ВВЕДЕНИЕ

Вывод на принтер несколько проще операций с экраном и диском. Для печати существует несколько операций, выполняющихся через DOS INT 21H и BIOS INT 17H. Команды, посылаемые на принтер, включают коды "конец страницы", "конец строки" и "возврат каретки".

Принтеры классифицируются по качеству печати. Матричный принтер формирует символы в виде матрицы точек и обеспечивает нормальный, узкий и широкий форматы символов. Более совершенные матричные принтеры обеспечивают точечную графику, наклонный шрифт, жирную печать и двойную плотность, а также могут печатать, например, символы игральных карт и другие алфавитно-цифровые символы. Высококачественные печатающие устройства ограничены набором символов на сменной "ромашке" или барабане, но обеспечивают отличное качество печати и большое разнообразие принтеров. Многие высококачественные принтеры обеспечивают печать 10, 12 или 15 символов на дюйм, а также пропорциональное расположение пробелов, подчеркивание, теневую и полужирную печать. Лазерные принтеры обладают преимуществами как для матричной графики, так и для качественной печати текстов.

Другая классификация печатающих устройств связана с интерфейсами. Компьютеры IBM PC имеют параллельный интерфейс, позволяющий передавать одновременно восемь битов из процессора на принтер. Кроме того, существует последовательный интерфейс, который выполняет побитовую передачу данных.

Многие принтеры имеют буфер памяти, объемом в несколько тысяч байтов. Принтеры также могут принимать биты контроля на четность (нечетность). Принтеры должны "понимать" специальные сигналы из процессора, например, для прогона листа, перевода строки или горизонтальной табуляции. В свою очередь, процессор должен "понимать" сигналы от принтера, указывающие на конец бумаги или состояние "занято".

К сожалению многие типы принтеров по разному реагируют на сигналы процессора и одной из наиболее сложных проблем для программистов - обеспечить соответствие собственных программ имеющимся печатающим устройством.

СИМВОЛЫ УПРАВЛЕНИЯ ПЕЧАТЬЮ

Стандартными символами управления печатью являются следующие:

Десятичн.	Шест.	Назначение
08	08	Возврат на шаг
09	09	Горизонтальная табуляция
10	0A	Перевод строки
11	0B	Вертикальная табуляция
12	0C	Прогон страницы
13	0D	Возврат каретки

Г о р и з о н т а л ь н а я т а б у л я ц и я. Горизонтальная табуляция (шест. 09) возможна только на принтерах, имеющих соответствующее обеспечение, иначе символы табуляции игнорируются. В последнем случае можно имитировать табуляцию выводом соответствующего числа пробелов.

П е р е в о д с т р о к и. Символ перевода строки (шест.OA) используется для прогона листа на один интервал. Соответственно для печати через два интервала используется два символа перевода строки.

П р о г о н с т р а н и ц ы. Установка бумаги после включения принтера определяет начальную позицию печати страницы. Длина страницы по умолчанию составляет 11 дюймов. Ни процессор, ни принтер автоматически не определяют конец страницы. Если ваша программа продолжает печатать после конца страницы, то произойдет переход через межстраничную перфорацию на на чало следующей страницы. Для управления страницами необходимо подсчитывать число напечатанных строк и при достижении максимального значения (например, 55 строк) выдать код прогона страницы (шест.0C) и, затем, сбросить счетчик строк в 0 или 1.

В конце печати необходимо выдать символ "перевода строки" или "прогона страницы" для вывода на печать данные последней строки, находящиеся в буфере печатающего устройства. Использование последнего символа "прогон страницы" позволяет установить напечатанный последний лист в положение для отрыва.

ФУНКЦИИ ПЕЧАТИ В РАСШИРЕННОЙ ВЕРСИИ DOS

В операционной системе DOS 2.0 имеются файловые указатели, которые были показаны в главах по управлению экраном дисплея и дисковой печати. Для вывода на печатающее устройство используется функция DOS шест.40 и стандартный файловый номер 04. Следующий пример демонстрирует печать 25 символов из области HEADG:

```
        HEADG     DB    'Industrial Bicycle Mfrs', 0DH, 0AH
                ...
                MOV  AH,40H    ;Запрос печати
                MOV  BX,04     ;Файловый номер принтера
                MOV  CX,25     ;25 символов
                LEA  DX,HEADG  ;Область вывода
                INT  21H       ;Вызов DOS
```

В случае ошибки операция устанавливает флаг CF и возвращает код ошибки в регистре AX.

ПРОГРАММА: ПОСТРАНИЧНАЯ ПЕЧАТЬ С ЗАГОЛОВКАМИ

```
TITLE   PRTNAME (COM)   Ввод и печать имен
CODESG  SEGMENT PARA PUBLIC 'CODE'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:CODESG
        ORG     100H
BEGIN:  JMP     SHORT MAIN
; ----------------------------------------------------
NAMEPAR LABEL   BYTE            ;Список параметров
MAXNLEN DB      20              ; максимальная длина имени
NAMELEN DB      ?               ; длина введенного имени
NAMEFLD DB      20 DUP(' ')     ; введенное имя
                                ;Строка заголовка:
HEADG   DB      'List of Employee Names    Page '
PAGECTR DB      '01',0AH,0AH
FFEED   DB      0CH             ;Перевод страницы
```

```
LFEED   DB      0AH             ;Перевод строки
LINECTR DB      01
PROMPT  DB      'Name? '
; ----------------------------------------------------
MAIN    PROC    NEAR
        CALL    Q10CLR          ;Очистить экран
        CALL    M10PAGE         ;Установка номера страницы
A2LOOP:
        MOV     DX,0000         ;Установить курсор в 00,00
        CALL    Q20CURS
        CALL    D10INPT         ;Ввести имя
        CALL    Q10CLR
        CMP     NAMELEN,00      ;Имя введено?
        JE      A30             ; если нет - выйти,
        CALL    E10PRNT         ; если да - подготовить
                                ;  печать
        JMP     A20LOOP
A30:
        MOV     CX,01           ;Конец работы:
        LEA     DX,FFEED        ; один символ
        CALL    P10OUT          ; для прогона страницы,
        RET                     ; возврат в DOS
MAIN    ENDP
;               Ввод имени с клавиатуры:
;               ----------------------
D10INPT PROC    NEAR
        MOV     AH,40H          ;Функция
        MOV     BX,01           ; вывода на экран
        MOV     CX,05           ; 5 символов
        LEA     DX,PROMPT
        INT     21H             ;Вызов DOS
        MOV     AH,0AH          ;Функция ввода с клавиатуры
        LEA     DX,NAMEPAR
        INT     21H             ;Вызов DOS
        RET
D10INPT ENDP
;               Подготовка для печати:
;               ----------------------
E10PRNT PROC    NEAR
        CMP     LINECTR,60      ;Конец страницы?
        JB      E20             ; нет - обойти
        CALL    M10PAGE         ; да - печатать заголовок
E20:    MOV     CH,00
        MOV     CL,NAMELEN      ;Число символов в имени
        LEA     DX,NAMEFLD      ;Адрес имени
        CALL    P10OUT          ;Печатать имя
        MOV     CX,01           ;Один
        LEA     DX,LFEED        ; перевод строки
        CALL    P10OUT
        INC     LINECTR         ;Увеличить счетчик строк
E10PRNT ENDP
;               Подпрограмма печати заголовка:
;               -----------------------------
M10PAGE PROC    NEAR
        CMP     WORD PTR PAGECTR,3130H  ;Первая страница?
        JE      M30             ; да - обойти
        MOV     CX,01           ;
        LEA     DX,FFEED        ; нет --
        CALL    P10OUT          ;  перевести страницу,
        MOV     LINECTR,03      ;  установить счетчик строк
M30:
```

```
       MOV     CX,36          ;Длина заголовка
       LEA     DX,HEADG       ;Адрес заголовка
M40:
       CALL    P10OUT
       INC     PAGECTR+1      ;Увеличить счетчик страниц
       CMP     PAGECTR+1,3AH  ;Номер страницы = шест.xx3A?
       JNE     M50            ; нет - обойти,
       MOV     PAGECTR+1,30H  ; да - перевести в ASCII
       INC     PAGECTR
M50:   RET
M10PAGE ENDP
;            Подпрограмма печати:
;            -------------------
P10OUT  PROC    NEAR          ;CX и DX установлены
       MOV     AH,40H         ;Функция печати
       MOV     BX,04          ;Номер устройства
       INT     21H            ;Вызов DOS
       RET
P10OUT  ENDP
;            Очистка экрана:
;            --------------
Q10CLR  PROC    NEAR
       MOV     AX,0600H       ;Функция прокрутки
       MOV     BH,60H         ;Цвет (07 для ч/б)
       MOV     CX,0000        ;От 00,00
       MOV     DX,184FH       ; до 24,79
       INT     10H            ;Вызов BIOS
       RET
Q10CLR  ENDP
;            Установка курсора (строка/столбец):
;            ---------------------------------
Q20CURS PROC    NEAR          ;DX уже установлен
       MOV     AH,02          ;Функция установки курсора
       MOV     BH,00          ;Страница № 0
       INT     10H            ;Вызов BIOS
       RET
Q20CURS ENDP
CODESG  ENDS
       END     BEGIN
```

Рис.19.1. Постраничная печать с заголовком.

Программа, приведенная на рис.19.1, аналогична программе на рис.9.1, за исключением того, что после ввода имен с клавиатуры выводит их не на экран, а на печатающее устройство. Каждая напечатанная страница содержит заголовок и через двойной интервал список введенных имен в следующем виде:

```
List of Employee Names   Page 01
Clancy Alderson
Ianet Brown
David Christie
...
```

Программа подсчитывает число напечатанных строк и при достижении конца страницы выполняет прогон до начала следующей страницы. В программе имеются процедуры:

D10INPT   Выдает на экран запрос и затем вводит имя с клавиатуры.
E10PRNT   Выводит имя на печатающее устройство (длина имени берется из

вводного списка параметров); в конце страницы вызывает процедуру M10PAGE.

M10PAGE Выполняет прогон на новую страницу, печатает заголовок, сбрасывает счетчик строк и увеличивает счетчик страниц на единицу.

P100UT Общая подпрограмма для непосредственного вывода на печатающее устройство.

В начале выполнения необходимо напечатать заголовок, но не делать перед этим перевод страницы. Поэтому процедура M10PAGE обходит перевод страницы, если счетчик PAGECTR содержит 01 (начальное значение). Поле PAGECTR определено как

```
PAGECTR  DB  '01'
```

В начале выполнения необходимо напечатать заголовок, но не делать перед этим перевод страницы. Поэтому процедура M10PAGE обходит перевод страницы, если счетчик PAGECTR содержит 01 (начальное значение). Поле PAGECTR определено как

```
PAGECTR   DB   '01'
```


В результате будет сгенерировано число в ASCII-коде - шест.3031. Процедура M10PAGE увеличивает счетчик PAGECTR на 1 так, что значение становится последовательно 3032, 3033 и т.д. Эти значения корректны до 3039, далее следует 303A, что будет распечатано, как двоеточие (:). Поэтому, если в правом байте поля PAGECTR появляется шест.3A, то это значение заменяется на шест.30, а к левому байту прибавляется единица. Таким образом шест.303A перекодируется в шест.3130, т.е. в 10 в символьном представлении.

Проверка на конец страницы до (но не после) печати имени гарантирует, что на последней странице будет напечатано по крайней мере одно имя под заголовком.

ПЕЧАТЬ ASCII-ФАЙЛОВ И ТАБУЛЯЦИЯ

Табуляция, обеспечиваемая, например, видеоадаптерами, заключается в замене одного символа табуляции (код 09) несколькими пробелами при выводе так, чтобы следующая позиция была кратна 8. Таким образом, стандартные позиции табуляции являются 8, 16, 24 и т.д. Многие принтеры, однако, игнорируют символы табуляции. Поэтому, такая программа, как DOS PRINT, предназначенная для печати ASCII файлов (например ассемблерных исходных текстов) проверяет каждый символ, посылаемый на принтер. И, если обнаруживается символ табуляции, то программа выдает несколько пробелов до позиции кратной 8.

Программа, приведенная на рис.19.2, выводит на экран запрос на ввод имени файла и, затем, печатает содержимое указанного файла. Эта программа в отличие от приведенной на рис.17.3 (вывод файлов на экран) осуществляет замену выводимых символов табуляции на соответствующее число пробелов. В результате символ табуляции в позициях от 0 до 7 приводит к переходу на позицию 8, от 8 до 15 - на 16 и т.д. Команды, реализующие данную логику, находятся в процедуре G10XFER после метки G60. Рассмотрим три примера обработки символа табуляции:

Текущая позиция печати:	1	9	21
Двоичное значение:	00000001	00001001	00010101
Очистка трех правых битов:	00000000	00001000	00010000
Прибавление 8:	00001000	00010000	00011000
Новая позиция:	8	16	24

В программе организованы следующие процедуры:

C10PRMP Запрашивает ввод имени файла. Нажатие только клавиши Return приводит к завершению работы программы.
E10OPEN Открывает дисковый файл по указанному имени.
G10XFER Контролирует конец сектора, конец файла, конец области вывода, символы "перевод строки" и табуляции. Пересылает обычные символы в область вывода.
P10PRNT Распечатывает выводную строку и очищает область вывода.
R10READ Считывает сектор из дискового файла.

Коды "возврат каретки", "перевод строки" и "прогон страницы" действительны для любых принтеров. Можно модифицировать программу для подсчета распечатываемых строк и выполнения прогона страницы (шест.0C) при достижении, например, строки 62.

Некоторые пользователи предпочитают устанавливать символы "прогон страницы" в ASCII файлах с помощью текстового редактора в конкретных местах текста, например, в конце ассемблерных процедур. Кроме того, можно изменить программу для функции 05 базовой версии DOS. Эта функция выполняет вывод каждого символа непосредственно на принтер. Таким образом можно исключить определение и использование области вывода.

```
TITLE   PRINASK (COM)   Чтение и печать дисковых записей
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:CODESG
        ORG     100H
BEGIN   JMP     MAIN
; --------------------------------------------------------
PATHPAR LABEL   BYTE            ;Список параметров для
MAXLEN  DB      32              ; ввода
NAMELEN DB      ?               ; имени файла
FILENAM DB      32 DUP(' ')
SECTOR  DB      512 DUP(' ')    ;Область ввода для файла
DISAREA DB      120 DUP(' ')    ;Область вывода
COUNT   DW      00
ENDCDE  DW      00
FFEED   DB      0CH
HANDLE  DW      0
OPENMSG DB      '*** Open error ***'
PROMPT  DB      'Name of file? '
; --------------------------------------------------------
MAIN    PROC    NEAR            ;Основная программа
        CALL    Q10SCR          ;Очистить экран
        CALL    Q20CURS         ;Установить курсор
A10LOOP:
        MOV     ENDCDE,00       ;Начальная установка
        CALL    C10PRMP         ;Получить имя файла
        CMP     NAMELEN,00      ;Есть запрос?
        JE      A90             ; нет - выйти
        CALL    E10OPEN         ;Открыть файл,
                                ; установить DTA
        CMP     ENDCDE,00       ;Ошибка при открытии?
        JNE     A80             ; да - повторить запрос
        CALL    R10READ         ;Прочитать первый сектор
        CMP     ENDCDE,00       ;Конец файла, нет данных?
        JE      A80             ; да - повторить запрос
        CALL    G10XPER         ;Распечатать сектор
A80:
        JMP     A10LOOP
```

```
A90:    RET
MAIN    ENDP
;             Подпрограмма запроса имени файла:
;             -------------------------------
C10PRMP PROC    NEAR
       MOV     AH,40H          ;Функция вывода на экран
       MOV     BX,01
       MOV     CX,13
       LEA     DX,PROMPT
       INT     21H
       MOV     AH,0AH          ;Функция ввода с клавиатуры
       LEA     DX,PATHPAR
       INT     21H
       MOV     BL,NAMELEN      ;Записать
       MOV     BH,00           ; 00 в конец
       MOV     FILENAM[BX],0   ; имени файла
C90     RET
C10PRMP ENDP
;             Открытие дискового файла:
;             -----------------------
E10OPEN PROC    NEAR
       MOV     AH,3DH          ;Функция открытия
       MOV     AL,00           ;Только чтение
       LEA     DX,FILENAM
       INT     21H
       JNC     E20             ;Проверить флаг CF
       CALL    X10ERR          ; ошибка, если установлен
       RET
E20:
       MOV     HANDLE,AX       ;Сохранить номер файла
       MOV     AX,2020H
       MOV     CX,256          ;Очистить пробелами
       REP     STOSW           ; область сектора
       RET
E100PEN ENDP
;             Подготовка и печать данных:
;             -------------------------
G10XFER PROC    NEAR
       CLD                     ;Направление слева-направо
       LEA     SI,SECTOR       ;Начальная установка
G20:
       LEA     DI,DISAREA
       MOV     COUNT,00
G30:
       LEA     DX,SECTOR+512
       CMP     SI,DX           ;Конец сектора?
       JNE     G40
       CALL    R10READ         ; да - читать следующий
       CMP     ENDCDE,00       ;Конец файла?
       JE      G80             ; да - выйти
       LEA     SI,SECTOR
G40:
       MOV     BX,COUNT
       CMP     BX,80           ;Конец области вывода?
       JB      G50             ; нет - обойти
       MOV     [DI+BX],0D0AH   ; да - записать CR/LF
       CALL    P10PRNT
       LEA     DI,DISAREA      ;Начало области вывода
G50:
       LODSB                   ;Записать [SI] в AL,
                               ; увеличить SI
```

```
        MOV     BX,COUNT
        MOV     [DI+BX],AL      ;Записать символ
        INC     BX
        CMP     AL,1AH          ;Конец файла?
        JE      G80             ; да - выйти
        CMP     AL,0AH          ;Конец строки?
        JNE     G60             ; нет - обойти,
        CALL    P10PRNT         ; да - печатать
        JMP     G20
G60:
        CMP     AL,09H          ;Символ табуляции?
        JNE     G70
        DEC     BX              ; да - установить BX:
        MOV     BYTE PTR [DI+BX],20H  ;Заменит TAB на пробел
        AND     BX,0FFF8H       ;Обнулить правые 8 бит
        ADD     BX,08           ; и прибавить 8
G70:
        MOV     COUNT,BX
        JMP     G30
G80:    MOV     BX,COUNT        ;Конец файла
        MOV     BYTE PTR [DI+BX],0CH    ;Прогон страницы
        CALL    P10PRNT         ;Печатать последнюю строку
G90:    RET
G10XFER ENDP
;               Подпрограммы печати:
;               --------------------
P10PRNT PROC    NEAR
        MOV     AH,40H          ;Функция печати
        MOV     BX,04
        MOV     CX,COUNT        ;Длина
        INC     CX
        LEA     DX,DISAREA
        INT     21H
        MOV     AX,2020H        ;Очистить область вывода
        MOV     CX,60
        LEA     DI,DISAREA
        REP     STOSW
        RET
P10PRNT ENDP
;               Подпрограмма чтения сектора:
;               ----------------------------
R10READ PROC    NEAR
        MOV     AH,3FH          ;Функция чтения
        MOV     BX,HANDLE       ;Номер файла
        MOV     CX,512          ;Длина
        MOV     DX,SECTOR       ;Буфер
        INT     21H
        MOV     ENDCDE,AX
        RET
R10READ ENDP
;               Прокрутка экрана:
;               ----------------
Q10SCR  PROC    NEAR
        MOV     AX,0600H
        MOV     BH,1EH          ;Установить цвет
        MOV     CX,0000         ;Прокрутка (сскроллинг)
        MOV     DX,184FH
        INT     10H
        RET
Q10SCR  ENDP
;               Подпрограмма установки курсора:
```

```
;              ------------------------------
Q20CURS PROC    NEAR
      MOV     AH,02          ;Функция установки
      MOV     BH,00          ; курсора
      MOV     DX,00
      INT     10H
      RET
Q20CURS ENDP
;              Вывод сообщения об ошибке:
;              -------------------------
X10ERR  PROC    NEAR
      MOV     AH,40H          ;Функция вывода на экран
      MOV     BX,01          ;Номер
      MOV     CX,18          ;Длина
      LEA     DX,OPENMSG      ;Адрес сообщения
      INT     1H

      MOV     NDCDE,01        ;Признак ошибки
      RET
X10ERR  ENDP
CODESG  ENDS
      END     BEGIN
```

Рис.19.2. Печать ASCII файла.

ПЕЧАТЬ ПОД УПРАВЛЕНИЕМ БАЗОВОЙ DOS

Для печати в базовой версии DOS необходимо установить в регистре AH код функции 05, а в регистр DL поместить распечатываемый символ и, затем, выполнить команду INT 21H следующим образом:

```
MOV  AH,05     ;Запрос функции печати
MOV  DL,char   ;Распечатываемый символ
INT  21H       ;Вызов DOS
```

С помощью этих команд можно передавать на принтер управляющие символы. Однако, печать, обычно, предполагает вывод полной или частичной строки текста и пошаговую обработку области данных, отформатированной по строкам.

Ниже показана программа печати полной строки. Сначала в регистр SI загружается начальный адрес области HEADG, а в регистр CX - длина этой области. Цикл, начинающийся по метке P20, выделяет очередной символ из области HEADG и посылает его на принтер. Так как первый символ области HEADG - "прогон страницы", а последние два - "перевод строки", то заголовок печатается в начале новой страницы и после него следует двойной интервал.

```
HEADG  DB   0CH,'Industrial Bicycle Mfrs',0DH,0AH,0AH
       LEA  SI,HEADG       ;Установка адреса и
       MOV  CX,27          ; длины заголовка
P20:
       MOV  AH,05          ;Запрос функции печати
       MOV  DL,[SI]        ;Символ из заголовка
       INT  21H            ;Вызов DOS
       INC  SI             ;Следующий символ
       LOOP P20
```

Пока принтер не включен, DOS выдает сообщения "Out of paper". После включения питания программа начинает работать нормально. Для прекращения

печати можно нажать клавиши Ctrl/Break.

СПЕЦИАЛЬНЫЕ КОМАНДЫ ПРИНТЕРА

Выше уже был показан ряд команд, которые являются основными для большинства печатающих устройств. Кроме того существуют следующие команды:

Десятичн.	Шест.	
15	0F	Включить узкий формат
14	0E	Включить широкий формат
18	12	Выключить узкий формат
20	14	Выключить широкий формат

Есть команды, которые распознаются по предшествующему символу Esc (шест.1B). Некоторые из них в зависимости от печатающего устройства представлены ниже:

```
1B 30   Установить плотность 8 строк на дюйм
1B 32   Установить плотность 6 строк на дюйм
1B 45   Включить жирный формат
1B 46   Выключить жирный формат
```

Коды команд можно посылать на принтер двумя разными способами:

1. Определить команды в области данных. Следующий пример устанавливает узкий формат, 8 строк на дюйм, затем печатает заголовок с завершающими командами "возврат каретки" и "перевод строки":

```
HEADG  DB  0FH, 1BH, 30H, 'Title...', 0DH, 0AH
```

2. Использовать команды с непосредственными данными:

```
MOV  AH,05        ;Запрос функции печати
MOV  DL,0FH       ;Включить узкий формат
INT  21H
```

Все последующие символы будут печататься в узком формате до тех пор, пока программа не выдаст на принтер команду, выключающую этот формат.

Приведенные команды не обязательно работают на принтерах любых моделей. Для проверки возможных команд управления следует ознакомиться с руководством конкретного печатающего устройства.

ПЕЧАТЬ С ПОМОЩЬЮ BIOS INT 17H

Прерывание BIOS INT 17H обеспечивает три различные операции, специфицированные содержимым регистра AH:

AH=0: Данная операция выполняет печать одного символа на три принтера по номерам 0,1 и 2 (стандартное значение - 0).

```
MOV  AH,00     ;Запрос функции печати
MOV  AL,char   ;Символ, выводимый на печать
MOV  DX,00     ;Выбор принтера № 0
INT  17H       ;Вызов BIOS
```

Если операция не может распечатать символ, то в регистре AH

устанавливается значение 01.

AH=1: Инициализация порта печатающего устройства:

```
MOV  AH,01     ;Запрос на инициализацию порта
MOV  DX,00     ;Выбор порта № 0
INT  17H       ;Вызов BIOS
```

Данная операция посылает на принтер символ "прогон страницы", поэтому ее можно использовать для установки положения "верх страницы". Большинство принтеров выполняют данную установку автоматически при включении.

AH=2: Чтение состояние порта принтера:

```
MOV  AH,02     ;Функция чтения состояния порта
MOV  DX,00     ;Выбор порта № 0
INT  17H       ;Вызов BIOS
TEST AH,00101001B;  Принтер готов?
JNZ  errormsg  ;Нет - выдать сообщение об ошибке
```

Назначение функций AH=1 и AH=2 состоит в определении состояния принтера. В результате выполнения этих функций биты регистра AH могут устанавливаться в 1:

Бит	Причина
7	Не занято
6	Подтверждение от принтера
5	Конец бумаги
4	Выбран
3	Ошибка ввода/вывода
0	Таймаут

Если принтер включен, то операция возвращает шест.90 или двоичное 10010000 - принтер "не занят" и "выбран" - это нормальное состояние готовности. В случае неготовности принтера устанавливаются бит 5 (конец бумаги или бит 3 (ошибка вывода). Если принтер выключен, то операция возвращает шест.B0 или двоичное 10110000, указывая на "конец бумаги".

Выполняя программу при выключенном принтере, BIOS не выдает сообщение автоматически, поэтому предполагается, что программа должна сама проверить и отреагировать на состояние принтера. Если программа не делает этого, то единственной индикацией будет мигающий курсор на экране дисплея. Если в этот момент включить принтер, то некоторые выходные данные могут быть потеряны. Следовательно, прежде чем использовать функции BIOS для печати, следует проверить состояние порта принтера и, если будет обнаружена ошибка, то выдать соответствующее сообщение. (Функции DOS выполняют эту проверку автоматически, хотя их сообщение "Out of paper" относится к различным состояниям). После включения принтера, вывод сообщений об ошибке прекращается и принтер начинает нормально работать без потери данных.

В процессе работы принтер может выйти за страницу или быть нечаянно выключен. Поэтому в программах печати следует предусмотреть проверку состояния принтера перед каждой попыткой печати.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Прежде чем выводить данные на печатающее устройство, включите принтер и вставьте в него бумагу.

- Для завершении печати используйте символы "перевод строки" и

"прогон страницы" для очистки буфера принтера.

- Функции DOS для печати предусматривают вывод сообщений при возникновении ошибки принтера. Функции BIOS возвращают только код состояния. При использовании BIOS INT 17H проверяйте состояние принтера перед печатью.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

19.1. Напишите программу в расширенной версии DOS для а) прогона страницы; б) печати вашего имени; в) перевода строки и печати вашего адреса; г) перевода строки и печати названия вашего города/штата (республики); д) прогона страницы.

19.2. Переделайте программу из предыдущего вопроса для базовой версии DOS.

19.3. Закодируйте строку, в которой имеется следующая информация: возврат каретки, прогон страницы, включение узких букв, заголовок (любое имя) и выключение узких букв.

19.4. Измените программу из вопроса 19.1 для использования BIOS INT 17H. Обеспечьте проверку состояния принтера.

19.5. Измените программу из вопроса 19.1 так, чтобы пункты б), в), г) выполнялись по 5 раз.

19.6. Измените программу на рис.19.1 для выполнения в базовой версии DOS.

19.7. Измените программу на рис.19.2 так, чтобы распечатываемые строки также выводились на экран.

ГЛАВА 20 Макросредства

Ц е л ь: Объяснить определение и использование ассемблерных макрокоманд.

ВВЕДЕНИЕ

Для каждой закодированной команды ассемблер генерирует одну команду на машинном языке. Но для каждого закодированного оператора компиляторного языка Pascal или C генерируется один или более (чаще много) команд машинного языка. В этом отношении можно считать, что компиляторный язык состоит из макрооператоров.

Ассемблер MASM также имеет макросредства, но макросы здесь определяются программистом. Для этого задается имя макроса, директива MACRO, различные ассемблерные команды, которые должен генерировать данный макрос и для завершения макроопределения - директива MEND. Затем в любом месте программы, где необходимо выполнение определенных в макрокоманде команд, достаточно закодировать имя макроса. В результате ассемблер сгенерирует необходимые команды.

Использование макрокоманд позволяет:

- упростить и сократить исходный текст программы;
- сделать программу более понятной;
- уменьшить число возможных ошибок кодирования.

Примерами макрокоманд могут быть операции ввода-вывода, связанные с инициализацией регистров и выполнения прерываний преобразования ASCII и двоичного форматов данных, арифметические операции над длинными полями, обработка строковых данных, деление с помощью вычитания.

В данной главе рассмотрены особенности макросредств, включая те, которые не достаточно ясно даны в руководстве по ассемблеру. Тем не менее пояснения для некоторых малоиспользуемых операций следует искать в руководстве по ассемблеру.

ПРОСТОЕ МАКРООПРЕДЕЛЕНИЕ

Макроопределение должно находиться до определения сегмента. Рассмотрим пример простого макроопределения по имени INIT1, которое инициализирует сегментные регистры для EXE-программы:

```
INIT1  MACRO                                   ;Начало
       ASSUME CS:CSEG,DS:DSEG,SS:STACK;ES:DSEG ;________________
       PUSH DS                          ;                |
       SUB  AX,AX                       ;                |
       PUSH AX                          ;Тело            |
       MOV  AX,DSEG                     ;макроопределения|
       MOV  DS,AX                       ;                |
       MOV  ES,AX                       ;________________|
       ENDM                             ;Конец
```

Директива MACRO указывает ассемблеру, что следующие команды до директивы ENDM являются частью макроопределения. Имя макрокоманды - INIT1, хотя здесь возможны другие правильные уникальные ассемблерные имена. Директива ENDM завершает макроопределение. Семь команд между директивами MACRO и ENDM составляют тело макроопределения.

Имена, на которые имеются ссылки в макроопределении, CSEG, DSEG и STACK должны быть определены где-нибудь в другом месте программы. Макрокоманда INIT1 может использоваться в кодовом сегменте там, где необходимо инициализировать регистры. Когда ассемблер анализирует команду INIT1, он сначала просматривает таблицу мнемокодов и, не обнаружив там соответствующего элемента, проверяет макрокоманды. Так как программа содержит определение макрокоманды INIT1 ассемблер подставляет тело макроопределения, генерируя необходимые команды - макрорасширение. Программа использует рассматриваемую макрокоманду только один раз, хотя имеются другие макрокоманды, предназначенные на любое число применений и для таких макрокоманд ассемблер генерирует одинаковые макрорасширения.

На рис.20.1 показана ассемблированная программа. В листинге макрорасширения каждая команда, помеченная слева знаком плюс (+), является результатом генерации макрокоманды. Кроме того, в макрорасширении отсутствует директива ASSUME, так как она не генерирует объектный код.

В последующем разделе "Включение из библиотеки макроопределений показана возможность каталогизации макрокоманд в библиотеке и автоматическое включение их в любые программы.

```
TITLE   MACRO1  (EXE) Макрос для инициализации
; ------------------------------------------
INIT1   MACRO
        ASSUME  CS:CSEG,DS:DSEG,SS:STACK,ES:DSEG
```

```
                  PUSH    DS
                  SUB     AX,AX
                  PUSH    AX
                  MOV     AX,DSEG
                  MOV     DS,AX
                  MOV     ES,AX
                  ENDM                    ;Конец макрокоманды
                  ; -----------------------------------------
0000                STACK   SEGMENT PARA STACK 'Stack'
0000     20 [ ???? ]         DW      32 DUP(?)
0040                STACK   ENDS
                  ; -----------------------------------------
0000                DSEG    SEGMENT PARA 'Data'
0000  54 65 73 74 20 6F   MESSGE  DB      'Test of macro-instruction', 13
    66 20 6D 61 63 72
    6F 2D 69 6E 73 74
    72 65 73 74 69 6F
    6E 0D
001A                DSEG    ENDS
                  ; -----------------------------------------
0000                CSEG    SEGMENT PARA 'Code'
0000                BEGIN   PROC    FAR
                       INIT1              ;Макрокоманда
0000  1E              +       PUSH    DS
0001  2B C0           +       SUB     AX,AX
0003  50              +       PUSH    AX
0004  B8 ---- R       +       MOV     AX,DSEG
0007  8E D8           +       MOV     DS,AX
0009  8E C0           +       MOV     ES,AX
000B  B4 40                   MOV     AH,40H     ;Вывод на экран
000D  BB 0001                 MOV     BX,01      ;Номер
0010  B9 001A                 MOV     CX,26      ;Длина
0013  8D 16 0000 R            LEA     DX,MESSGE  ;Сообщение
0017  CD 21                   INT     21H
0019  CB                      RET
001A                BEGIN   ENDP
001A                CSEG    ENDS
                       END     BEGIN

Macros:
              N a m e         Length
INIT1. . . . . . . . . . . . . . . . 0004

Segments and Groups:
              N a m e         Size    Align   Combine  Class
CSEG . . . . . . . . . . . . . . . . 001A    PARA    NONE     'CODE'
DSEG . . . . . . . . . . . . . . . . 001A    PARA    NONE     'DATA'
STACK. . . . . . . . . . . . . . . . 0040    PARA    STACK    'STACK'

Symbols:
              N a m e         Type    Value   Attr
BEGIN. . . . . . . . . . . . . . . . F PROC  0000    CSEG    Length=001A
MESSAGE. . . . . . . . . . . . . . . L BYTE  0000    DSEG
```

Рис.20.1. Пример ассемблирования макрокоманды.

ИСПОЛЬЗОВАНИЕ ПАРАМЕТРОВ В МАКРОКОМАНДАХ

В предыдущем макроопределении требовались фиксированные имена сегментов: CSEG, DSEG и STACK. Для того, чтобы макрокоманда была более гибкой и могла принимать любые имена сегментов, определим эти имена, как формальные параметры:

```
INIT2  MACRO CSNAME,DSNAME,SSNAME   ;Формальные параметры
       ASSUME CS:CSNAME,DS:DSNAME,SC:SSNAME,ES:DSNAME
       PUSH DS
       SUB  AX,AX
       PUSH AX
       MOV  AX,DSNAME
       MOV  DS,AX
       MOV  ES,AX
       ENDM                         ;Конец макроопределения
```

Формальные параметры в макроопределении указывают ассемблеру на соответствие их имен любым аналогичным именам в теле макроопределения. Все три формальных параметра CSNAME, DSNAME и SSNAME встречаются в директиве ASSUME, а параметр DSNAME еще и в последующей команде MOV. Формальные параметры могут иметь любые правильные ассемблерные имена, не обязательно совпадающими именами в сегменте данных.

Теперь при использовании макрокоманды INIT2 необходимо указать в качестве параметров действительные имена трех сегментов в соответствующей последовательности. Например, следующая макрокоманда содержит три параметра, которые соответствуют формальным параметрам в исходном макроопределении:

```
Макроопределение: INIT2  MACRO  CSNAME,DSNAME,SSNAME (форм. параметры)
                                |      |      |
Макрокоманда:            INIT2  CSEG,DSEG,STACK (параметры)
```

Так как ассемблер уже определил соответствие между формальными параметрами и операторами в макроопределении, то теперь ему остается подставить параметры макрокоманды в макрорасширении:

- Параметр 1: CSEG ставится в соответствие с CSNAME в макроопределении. Ассемблер подставляет CSEG вместо CSNAME в директиве ASSUME.
- Параметр 2: DSEG ставится в соответствие с DSNAME в макроопределении. Ассемблер подставляет DSEG вместо двух DSNAME: в директиве ASSUME и в команде MOV.
- Параметр 3: STACK ставится в соответствие с SSNAME в макроопределении. Ассемблер подставляет STACK вместо SSNAME в директиве ASSUME.

Макроопределение с формальными параметрами и соответствующее макрорасширение приведены на рис.20.2.

Формальный параметр может иметь любое правильное ассемблерное имя (включая имя регистра, например, CX), которое в процессе ассемблирования будет заменено на параметр макрокоманды. Отсюда следует, что ассемблер не распознает регистровые имена и имена, определенные в области данных, как таковые. В одной макрокоманде может быть определено любое число формальных параметров, разделенных запятыми, вплоть до 120 колонки в строке.

```
TITLE   MACRO2  (EXE) Использование параметров
; --------------------------------------------
INIT2   MACRO   CSNAME,DSNAME,SSNAME
        ASSUME  CS:CSNAME,DS:DSNAME
        ASSUME  SS:SSNAME,ES:DSNAME
```

```
                         PUSH    DS
                         SUB     AX,AX
                         PUSH    AX
                         MOV     AX,DSNAME
                         MOV     DS,AX
                         MOV     ES,AX
                         ENDM              ;Конец макрокоманды
                   ; -------------------------------------------
0000                 STACK   SEGMENT PARA STACK 'Stack'
0000     20 [ ???? ]         DW      32 DUP(?)
0040                 STACK   ENDS
                   ; -------------------------------------------
0000                 DSEG    SEGMENT PARA 'Data'
0000  54 65 73 74 20 6F  MESSAGE DB      'Test of macro', '$'
      66 20 6D 61 63 72
      6F 24
000E                 DSEG    ENDS
                   ; -------------------------------------------
0000                 CSEG    SEGMENT PARA 'Code'
0000                 BEGIN   PROC    FAR
                         INIT2   CSEG,DSEG,STACK
0000  1E              +        PUSH    DS
0001  2B C0           +        SUB     AX,AX
0003  50              +        PUSH    AX
0004  B8 ---- R       +        MOV     AX,DSEG
0007  8E D8           +        MOV     DS,AX
0009  8E C0           +        MOV     ES,AX
000B  B4 09                    MOV     AH,09        ;Вывод на экран
000D  8D 16 0000 R             LEA     DX,MESSGE    ;Сообщение
0011  CD 21                    INT     21H
0013  CB                       RET
0014                 BEGIN   ENDP
0014                 CSEG    ENDS
                         END     BEGIN
```

Рис.20.2. Использование параметров в макрокомандах.

КОММЕНТАРИИ

Для пояснений назначения макроопределения в нем могут находиться комментарии. Директива COMMENT или символ точка с запятой указывают на строку комментария, как это показано в следующем макроопределении PROMPT:

```
        PROMPT    MACRO    MESSGE
        ;Эта макрокоманда выводит сообщения на экран
                MOV      AH,09H
                LEA      DX,MESSGE
                INT      21H
                ENDM
```

Так как по умолчанию в листинг попадают только команды генерирующие объектный код, то ассемблер не будет автоматически выдавать и комментарии, имеющиеся в макроопределении. Если необходимо, чтобы в расширении появлялись комментарии, следует использовать перед макрокомандой директиву .LALL ("list all" - выводить все), которая кодируется вместе с лидирующей точкой:

```
        .LALL
        PROMPT    MESSAG1
```

Макроопределение может содержать несколько комментариев, причем некоторые из них могут выдаваться в листинге, а другие - нет. В первом случае необходимо использовать директиву .LALL. Во втором - кодировать перед комментарием два символа точка с запятой (;;) - признак подавления вывода комментария в листинг. По умолчанию в ассемблере действует директива .XALL, которая выводит в листинг только команды, генерирующие объектный код. И, наконец, можно запретить появление в листинге ассемблерного кода в макрорасширениях, особенно при использовании макрокоманды в одной программе несколько раз. Для этого служит директива .SALL ("suppress all" - подавить весь вывод), которая уменьшает размер выводимого листинга, но не оказывает никакого влияния на размер объектного модуля.

Директивы управления листингом .LALL, .XALL, .SALL сохраняют свое действие по всему тексту программы, пока другая директива листинга не изменит его. Эти директивы можно размещать в программе так, чтобы в одних макрокомандах распечатывались комментарии, в других - макрорасширения, а в третьих подавлялся вывод в листинг.

Программа на рис.20.3 демонстрирует описанное выше свойство директив листинга. В программе определено два макроопределения INIT2 и PROMPT, рассмотренные ранее. Кодовый сегмент содержит директиву .SALL для подавления распечатки INIT2 и первого расширения PROMPT. Для второго расширения PROMPT директива .LALL указывает ассемблеру на вывод в листинг комментария и макрорасширения. Заметим, однако, что комментарий, отмеченный двумя символами точка с запятой (;;) в макроопределении PROMPT, не распечатывается в макрорасширениях независимо от действия директив управления листингом.

```
                    TITLE   MACRO3  (EXE) Директивы .LALL и .SALL
                    ; ---------------------------------------------
                    INIT2   MACRO   CSNAME,DSNAME,SSNAME
                          ASSUME  CS:CSNAME,DS:DSNAME
                          ASSUME  SS:SSNAME,ES:DSNAME
                          PUSH    DS
                          SUB     AX,AX
                          PUSH    AX
                          MOV     AX,DSNAME
                          MOV     DS,AX
                          MOV     ES,AX
                          ENDM
                    ; ---------------------------------------------
                    PROMPT  MACRO   MESSAGE
                    ;   Макрокоманда выводит на экран любые сообщения
                    ;;  Генерирует команды вызова DOS
                          MOV     AH,09         ;Вывод на экран
                          LEA     DX,MESSAGE
                          INT     21H
                          ENDM
                    ; ---------------------------------------------
0000                  STACK   SEGMENT PARA STACK 'Stack'
0000     20 [ ???? ]          DW      32 DUP (?)
0040                  STACK   ENDS
                    ; ---------------------------------------------
0000                  DATA    SEGMENT PARA 'Data'
0000  43 75 73 74 6F 6D  MESSG1  DB      'Customer name?', '$'
     65 72 20 6E 61 6D
     65 3F 24
000F  43 75 73 74 6F 6D  MESSG2  DB      'Customer address?', '$'
     65 72 20 61 64 64
```

```
     72 65 73 73 3F 24
0021                DATA    ENDS
                  ; -----------------------------------------------
0000                CSEG    SEGMENT PARA 'Code'
0000                BEGIN   PROC    FAR
                        .SALL
                        INIT2   CSEG,DATA,STACK
                        PROMPT  MESSG1
                        .LALL
                        PROMPT  MESSG2
                 + ;   Макрокоманда выводит на экран любые сообщения
0013  B4 09           +       MOV     AH,09        ;Вывод на экран
0015  8D 16 000F R    +       LEA     DX,MESSG2
0019  CD 21           +       INT     21H
001B  CB                      RET
001C                BEGIN   ENDP
001C                CSEG    ENDS
                        END     BEGIN
```

Рис.20.3. Распечатка и подавление макрорасширений в листинге.

ИСПОЛЬЗОВАНИЕ МАКРОКОМАНД В МАКРООПРЕДЕЛЕНИЯХ

Макроопределение может содержать ссылку на другое макроопределение. Рассмотрим простое макроопределение DOS21, которое заносит в регистр AH номер функции DOS и выполняет INT 21H:

```
DOS21     MACRO     DOSFUNC
          MOV       AH,DOSFUNC
          INT       21H
          ENDM
```

Для использования данной макрокоманды при вводе с клавиатуры необходимо закодировать:

```
LEA       DX,NAMEPAR
DOS21     0AH
```

Предположим, что имеется другое макроопределение, использующее функцию 02 в регистре AH для вывода символа:

```
DISP      MACRO     CHAR
          MOV       AH,02
          MOV       DL,CHAR
          INT       21H
          ENDM
```

Для вывода на экран, например, звездочки достаточно закодировать макрокоманду DISP '*'. Можно изменить макроопределение DISP, воспользовавшись макрокомандой DOC21:

```
DISP      MACRO     CHAR
          MOV       DL,CHAR
          DOS21     02
          ENDM
```

Теперь, если закодировать макрокоманду DISP в виде DISP '*', то ассемблер сгенерирует следующие команды:

```
        MOV     DL,'*'
        MOV     AH,02
        INT     21H
```

ДИРЕКТИВА LOCAL

В некоторых макрокомандах требуется определять элементы данных или метки команд. При использовании такой макрокоманды в программе более одного раза происходит также неоднократное определение одинаковых полей данных или меток. В результате ассемблер выдаст сообщения об ошибке из-за дублирования имен. Для обеспечения уникальности генерируемых в каждом макрорасширении имен используется директива LOCAL, которая кодируется непосредственно после директивы MACRO, даже перед комментариями. Общий формат имеет следующий вид:

```
LOCAL dummy-1,dummy-2,...     ;Формальные параметры
```

Рис.20.4 иллюстрирует использование директивы LOCAL. В приведенной на этом рисунке программе выполняется деление вычитанием; делитель вычитается из делимого и частное увеличивается на 1 до тех пор, пока делимое больше делителя. Для данного алгоритма необходимы две метки: COMP - адрес цикла, OUT - адрес выхода из цикла по завершению. Обе метки COMP и OUT определены как LOCAL и могут иметь любые правильные ассемблерные имена.

В макрорасширении для COMP генерируется метка ??0000, а для OUT - ??0001. Если макрокоманда DIVIDE будет использована в этой программе еще один раз, то в следующем макрорасширении будут сгенерированы метки ??0002 и ??0003 соответственно. Таким образом, с помощью директивы LOCAL обеспечивается уникальность меток в макрорасширениях в одной программе.

```
                TITLE   MACRO4  (COM) Использование директивы LOCAL
                ; -----------------------------------------------
                DIVIDE  MACRO   DIVIDEND,DIVISOR,QUOTIENT
                        LOCAL   COMP
                        LOCAL   OUT
                ;       AX=делимое, BX=делитель, CX=частное
                        MOV     AX,DIVIDEND    ;Загрузить делимое
                        MOV     BX,DIVISOR     ;Загрузить делитель
                        SUB     CX,CX          ;Регистр для частного
                COMP:
                        CMP     AX,BX          ;Делимое < делителя?
                        JB      OUT            ; да - выйти
                        SUB     AX,BX          ;Делимое - делитель
                        INC     CX             ;Частное + 1
                        JMP     COMP
                OUT:
                        MOV     QUOTIENT,CX    ;Записать результат
                        ENDM
                ; -----------------------------------------------
0000              CSEG    SEGMENT PARA 'Code'
                        ASSUME  CS:CSEG,DS:CSEG,SS:CSEG,ES:CSEG
0100                      ORG     100H
0100  EB 06             BEGIN:  JMP     SHORT MAIN
                ; -----------------------------------------------
0102  0096              DIVDND  DW      150         ;Делимое
0104  001B              DIVSOR  DW      27          ;Делитель
0106  ????             QUOTNT  DW      ?           ;Частное
```

```
                ; ---------------------------------------------
0108              MAIN    PROC    NEAR
                      .LALL
                      DIVIDE  DIVDND,DIVSOR,QUOTNT
               + ;       AX=делимое, BX=делитель, CX=частное
0108  A1 0102 R     +         MOV     AX,DIVDND      ;Загрузить делимое
010B  8B 1E 0104 R  +         MOV     BX,DIVSOR      ;Загрузить делитель
010F  2B C9         +         SUB     CX,CX          ;Регистр для частного
0111               + ??0000:
0111  3B C3         +         CMP     AX,BX          ;Делимое < делителя?
0113  72 05         +         JB      ??0001         ; да - выйти
0115  2B C3         +         SUB     AX,BX          ;Делимое - делитель
0117  41            +         INC     CX             ;Частное + 1
0118  EB F7         +         JMP     ??0000
011A               + ??0001:
011A  89 0E 0106 R  +         MOV     QUOTNT,CX      ;Записать результат
011E  C3                      RET
011F              MAIN    ENDP
011F              CSEG    ENDS
                      END     BEGIN
```

Рис.20.4. Использование директивы LOCAL.

ИСПОЛЬЗОВАНИЕ БИБЛИОТЕК МАКРООПРЕДЕЛЕНИЙ

Определение таких макрокоманд, как INIT1 и INIT2 и одноразовое их использование в программе кажется бессмысленным. Лучшим подходом здесь является каталогизация собственных макрокоманд в библиотеке на магнитном диске, используя любое описательное имя, например, MACRO.LIB:

```
INIT      MACRO     CSNAME,DSNAME,SSNAME
          .
          .
          ENDM
PROMPT    MACRO     MESSGE
          .
          .
          ENDM
```

Теперь для использования любой из каталогизированных макрокоманд вместо MACRO определения в начале программы следует применять директиву INCLUDE:

```
          INCLUDE   C:MACRO.LIB
          .
          .
          INIT      CSEG,DATA,STACK
```

В этом случае ассемблер обращается к файлу MACRO.LIB (в нашем примере) на дисководе C и включает в программу оба макроопределения INIT и PROMPT. Хотя в нашем примере требуется только INIT. Ассемблерный листинг будет содержать копию макроопределения, отмеченного символом C в 30 колонке LST-файла. Следом за макрокомандой идет ее расширение с объектным кодом и с символом плюс (+) в 31 колонке.

Так как транслятор с ассемблера является двухпроходовым, то для обеспечения обработки директивы INCLUDE только в первом проходе (а не в обоих) можно использовать следующую конструкцию:

```
IF1
```

```
                    INCLUDE   C:MACRO.LIB
            ENDIF
```

IF1 и ENDIF являются условными директивами. Директива IF1 указывает ассемблеру на необходимость доступа к библиотеке только в первом проходе трансляции. Директива ENDIF завершает IF-логику. Таким образом, копия макроопределений не появится в листинге - будет сэкономлено и время и память.

Программа на рис.20.5 содержит рассмотренные выше директивы IF1, INCLUDE и ENDIF, хотя в LST-файл ассемблер выводит только директиву ENDIF. Обе макрокоманды в кодовом сегменте INIT и PROMPT закаталогизированы в файле MACRO.LIB, т.е. просто записаны друг за другом на дисковый файл по имени MACRO.LIB с помощью текстового редактора.

Расположение директивы INCLUDE не критично, но она должна появиться ранее любой макрокоманды из включаемой библиотеки.

```
                    TITLE   MACRO5  (EXE) Проверка директивы INCLUDE
                    EDIF
                    ; ---------------------------------------------
0000                  STACK   SEGMENT PARA STACK 'Stack'
0000     20 [????]            DW      32 DUP(?)
0040                  STACK   ENDS
                    ; ---------------------------------------------
0000                  DATA    SEGMENT PARA 'Data'
0000  54 65 73 74 20 6F  MESSGE  DB      'Test of macro','$'
     66 20 6D 61 63 72
     6F 24
000E                  DATA    ENDS
                    ; ---------------------------------------------
0000                  CSEG    SEGMENT PARA 'Code'
0000                  BEGIN   PROC    FAR
                          INIT    CSEG,DATA,STACK
0000  1E              +        PUSH    DS
0001  3B  C0          +        SUB     AX,AX
0003  50              +        PUSH    AX
0004  B8 ---- R       +        MOV     AX,DATA
0007  8E D8           +        MOV     DS,AX
0009  8E C0           +        MOV     ES,AX
                          PROMPT  MESSGE
000B  B4 09           +        MOV     AH,09         ;Вывод на экран
000D  8D 16 0000 R    +        LEA     DX,MESSGE
0011  CD 21           +        INT     21H
0013  CB                       RET
0014                  BEGIN   ENDP
0014                  CSEG    ENDS
                          END     BEGIN
```

Рис.20.5. Использование библиотеки макроопределений.

Директива очистки

Директива INCLUDE указывает ассемблеру на включение всех макроопределений из специфицированной библиотеки. Например, библиотека содержит макросы INIT, PROMPT и DIVIDE, хотя программе требуется только INIT. Директива PURGE позволяет "удалить" нежелательные макросы PROMPT и DIVIDE в текущем ассемблировании:

```
IF1
        INCLUDE MACRO.LIB   ;Включить всю библиотеку
ENDIF
PURGE     PROMRT,DIYIDE       ;Удалить ненужные макросы
...
INIT      CSEG,DATA,STACK     ;Использование оставшейся макрокоманды
```

Директива PURGE действует только в процессе ассемблирования и не оказывает никакого влияния на макрокоманды, находящиеся в библиотеке.

КОНКАТЕНАЦИЯ (&)

Символ амперсанд (&) указывает ассемблеру на сцепление (конкатенацию) текста или символов. Следующая макрокоманда MOVE генерирует команду MOVSB или MOVSW:

```
        MOVE      MACRO     TAG
                REP       MOVS&TAG
                ENDM
```

Теперь можно кодировать макрокоманду в виде MOVE B или MOVE W. В результате макрорасширения ассемблер сцепит параметр с командой MOVS и получит REP MOVSB или REP MOVSW. Данный пример весьма тривиален и служит лишь для иллюстрации.

ДИРЕКТИВЫ ПОВТОРЕНИЯ: REPT, IRP, IRPC

Директивы повторения заставляют ассемблер повторить блок операторов, завершаемых директивой ENDM. Эти директивы не обязательно должны находится в макроопределении, но если они там находятся, то одна директива ENDM требуется для завершения повторяющегося блока, а вторая ENDM - для завершения макроопределения.

REPT: Повторение

Операция REPT приводит к повторению блока операторов до директивы ENDM в соответствии с числом повторений, указанным в выражении:

```
        REPT выражение
```

В следующем примере происходит начальная инициализация значения N=0 и затем повторяется генерация DB N пять раз:

```
        N =  0
        REPT 5
        N =  N + 1
        DB   N
        ENDM
```

В результате будут сгенерированы пять операторов DB от DB 1 до DB 5. Директива REPT может использоваться таким образом для определения таблицы или части таблицы. Другим примером может служить генерация пяти команд MOVSB, что эквивалентно REP MOVSB при содержимом CX равном 05:

```
        REPT 5
        MOVSB
```

```
        ENDM
```

IRP: Неопределенное повторение

Операция IRP приводит к повторению блока команд до директивы ENDM. Основной формат:

```
        IRP  dummy,<arguments>
```

Аргументы, содержащиеся в угловых скобках, представляют собой любое число правильных символов, строк, числовых или арифметических констант. Ассемблер генерирует блок кода для каждого аргумента. В следующем примере ассемблер генерирует DB 3, DB 9, DB 17, DB 25 и DB 28:

```
        IRP  N,<3, 9, 17, 25, 28>
        DB   N
        ENDM
```

IRPC: Неопределенное повторение символа

Операция IRPC приводит к повторению блока операторов до директивы ENDM. Основной формат:

```
        IRPC dummy,string
```

Ассемблер генерирует блок кода для каждого символа в строке "string". В следующем примере ассемблер генерирует DW 3, DW 4 ... DW 8:

```
        IRPC N,345678
        DW   N
        ENDM
```

УСЛОВНЫЕ ДИРЕКТИВЫ

Ассемблер поддерживает ряд условных директив. Ранее нам уже приходилось использовать директиву IF1 для включения библиотеки только в первом проходе ассемблирования. Условные директивы наиболее полезны внутри макроопределений, но не ограничены только этим применением. Каждая директива IF должна иметь спаренную с ней директиву ENDIF для завершения IF-логики и возможную директиву ELSE для альтернативного действия:

```
        IFxx    (условие)
        .                          ___________
        .                           Условный|
        ELSE (не обязательное действие)      |
        .                             блок  |
        .                          ___________|
        ENDIF    (конец IF-логики)
```

Отсутствие директивы ENDIF вызывает сообщение об ошибке: "Undeterminated conditional" (незавершенный условный блок). Если проверяемое условие истинно, то ассемблер выполняет условный блок до директивы ELSE или при отсутствии ELSE - до директивы ENDIF. Если условие ложно, то ассемблер выполняет условный блок после директивы ELSE, а при отсутствии ELSE вообще обходит условный блок.

Ниже перечислены различные условные директивы:

IF выражение — Если выражение не равно нулю, ассемблер обрабатывает операторы в условном блоке.

IFE выражение — Если выражение равно нулю, ассемблер обрабатывает операторы в условном блоке.

IF1 (нет выражения) — Если осуществляется первый проход ассемблирования то обрабатываются операторы в условном блоке.

IF2 (нет выражения) — Если осуществляется второй проход операторы ассемблирования, то обрабатываются в условном блоке.

IFDEF идентификатор — Если идентификатор определен в программе или объявлен как EXTRN, то ассемблер обрабатывает операторы в условном блоке.

IFNDEF идентификатор — Если идентификатор не определен в программе или не объявлен как EXTRN, то ассемблер обрабатывает операторы в условном блоке.

IFB <аргумент> — Если аргументом является пробел, ассемблер обрабатывает операторы в условном блоке. Аргумент должен быть в угловых скобках.

IFNB <аргумент> — Если аргументом является не пробел, то ассемблер обрабатывает операторы в условном блоке. Аргумент должен быть в угловых скобках.

IFIDN <арг-1>,<арг-2> — Если строка первого аргумента идентична строке второго аргумента, то ассемблер обрабатывает операторы в условном блоке. Аргументы должны быть в угловых скобках.

IFDIF<арг-1>,<арг-2> — Если строка первого аргумента отличается от строки второго аргумента, то ассемблер обрабатывает операторы в условном блоке. Аргументы должны быть в угловых скобках.

Ниже приведен простой пример директивы IFNB (если не пробел). Для DOS INT 21H все запросы требуют занесения номера функции в регистр AH, в то время как лишь некоторые из них используют значение в регистре DX. Следующее макроопределение учитывает эту особенность:

```
DOS21   MACRO   DOSFUNC,DXADDRES
        MOV     AN,DOSFUNC
        IFNB    <DXADDRES>
        MOV     DX,OFFSET DXADDRES
        ENDIF
        INT     21H
        ENDM
```

Использование DOS21 для простого ввода с клавиатуры требует установки значения 01 в регистр AH:

```
        DOS21   01
```

Ассемблер генерирует в результате команды MOV AH,01 и INT 21H. Для ввода символьной строки требуется занести в регистр AH значение 0AH, а в регистр DX - адрес области ввода:

```
        DOS21   0AH,IPFIELD
```

Ассемблер генерирует в результате обе команды MOV и INT 21H.

ДИРЕКТИВА ВЫХОДА ИЗ МАКРОСА EXITM.

Макроопределение может содержать условные директивы, которые

проверяют важные условия. Если условие истинно, то ассемблер должен прекратить дальнейшее макрорасширение. Для этой цели служит директива EXITM:

```
IFxx     [условие]
.
.       (неправильное условие)
.
EXITM
.
.
ENDIF
```

Как только ассемблер попадает в процессе генерации макрорасширения на директиву EXITM, дальнейшее расширение прекращается и обработка продолжается после директивы ENDM. Можно использовать EXITM для прекращения повторений по директивам REPT, IRP и IRPC даже если они находятся внутри макроопределения.

МАКРОКОМАНДЫ, ИСПОЛЬЗУЮЩИЕ IF И IFNDEF УСЛОВИЯ

Программа на рис.20.6 содержит макроопределение DIVIDE, которая генерирует подпрограмму для выполнения деления вычитанием. Макрокоманда должна кодироваться с параметрами в следующей последовательности: делимое, делитель, частное. Макрокоманда содержит директиву IFNDEF для проверки наличия параметров. Для любого неопределенного элемента макрокоманда увеличивает счетчик CNTR. Этот счетчик может иметь любое корректное имя и предназначен для временного использования в макроопределении. После проверки всех трех параметров, макрокоманда проверяет CNTR:

```
IF   CNTR
;Макрорасширение прекращено
EXITM
```

Если счетчик CNTR содержит ненулевое значение, то ассемблер генерирует комментарий и прекращает по директиве EXITM дальнейшее макрорасширение. Заметим, что начальная команда устанавливает в счетчике CNTR нулевое значение и, кроме того, блоки IFNDEF могут устанавливать в CNTR единичное значение, а не увеличивать его на 1.

Если ассемблер успешно проходит все проверки, то он генерирует макрорасширение. В кодовом сегменте первая макрокоманда DIVIDE содержит правильные делимое и частное и, поэтому генерирует только комментарии. Один из способов улучшения рассматриваемой макрокоманды - обеспечить проверку на ненулевой делитель и на одинаковый знак делимого и делителя; для этих целей лучше использовать коды ассемблера, чем условные директивы.

```
TITLE   MACRO6 (COM) Проверка директ. IF и IFNDEF
; ----------------------------------------------
DIVIDE  MACRO   DIVIDEND,DIVISOR,QUOTIENT
        LOCAL   COMP
        LOCAL   OUT
        CNTR    = 0
;       AX-делимое, BX-делитель, CX-частное
        IFNDEF  DIVIDEND
;               Делитель не определен
        CNTR    = CNTR +1
        ENDIF
```

```
                   IFNDEF  DIVISOR
               ;              Делимое не определено
                   CNTR    = CNTR +1
                   ENDIF
                   IFNDEF  QUOTIENT
               ;              Частное не определено
                   CNTR    = CNTR +1
                   ENDIF
                   IF      CNTR
               ;              Макрорасширение отменено
                   EXITM
                   ENDIF
                   MOV     AX,DIVIDEND   ;Загрузка делимого
                   MOV     BX,DIVISOR    ;Загрузка делителя
                   SUB     CX,CX       ;Регистр для частного
               COMP:
                   CMP      AX,BX      ;Делимое < делителя?
                   JB       OUT        ; да - выйти
                   SUB      AX,BX       ;Делимое - делитель
                   INC      CX          ;Частное + 1
                   JMP      COMP
               OUT:
                   MOV      QUOTIENT,CX  ;Запись результата
                   ENDM
               ; ---------------------------------------------
0000             CSEG    SEGMENT  PARA 'Code'
                   ASSUME   CS:CSEG,DS:CSEG,SS:CSEG,ES:CSEG
0100                 ORG      100H
0100  EB 06         BEGIN:  JMP      SHORT MAIN
               ; ---------------------------------------------
0102  0096          DIVDND  DW       150
0104  001B          DIVSOR  DW       27
0106  ????         QUOTNT  DW       ?
               ; ---------------------------------------------
0108             MAIN    PROC     NEAR
                   .LALL
                   DIVIDE   DIVDND,DIVSOR,QUOTNT
= 0000           +        CNTR    = 0
              + ;       AX-делимое, BX-делитель, CX-частное
              +         ENDIF
              +         ENDIF
              +         ENDIF
              +         ENDIF
0108  A1 0102 R       +        MOV     AX,DIVDND     ;Загрузка делимого
0108  8B 1E 0104 R     +        MOV     BX,DIVSOR     ;Загрузка делителя
010F  2B C9           +        SUB     CX,CX       ;Регистр для частного
0111              + ??0000:
0111  3B C3           +        CMP     AX,BX       ;Делимое < делителя?
0113  72 05           +        JB      ??0001      ; да - выйти
0115  2B C3           +        SUB     AX,BX        ;Делимое - делитель
0117  41             +        INC     CX
0118  EB F7           +        JMP     ??0000
011A              + ??0001:
011A  89 0E 0106 R     +        MOV     QUOTNT,CX     ;Запись результата
                   DIVIDE  DIDND,DIVSOR,QUOT
= 0000           +        CNTR    = 0
              + ;       AX-делимое, BX-делитель, CX-частное
              +         IFNDEF  DIDND
              + ;              Делитель не определен
= 0001           +        CNTR    = CNTR +1
              +         ENDIF
```

```
                  +        ENDIF
                  +        IFNDEF  QUOT
                  + ;            Частное не определено
= 0002               +        CNTR    = CNTR +1
                  +        ENDIF
                  +        IF      CNTR
                  + ;            Макрорасширение отменено
                  +        EXITM
011E  C3                     RET
011F                  MAIN    ENDP
011F                  CSEG    ENDS
                          END    BEGIN
```

Рис.20.6. Использование директив IF и IFNDEF.

МАКРОС, ИСПОЛЬЗУЮЩИЙ IFIDN-УСЛОВИЕ

Программа на рис.20.7 содержит макроопределение по имени MOVIF, которая генерирует команды MOVSB или MOVSW в зависимости от указанного параметра. Макрокоманду можно кодировать с параметром B (для байта) или W (для слова) для генерации команд MOVSB или MOVSW из MOVS.

Обратите внимание на первые два оператора в макроопределении:

```
        MOVIF     MACRO     TAG
                IFIDN     <&TAG>,<B>
```

Условная директива IFIDN сравнивает заданный параметр (предположительно B или W) со строкой B. Если значения идентичны, то ассемблер генерирует REP MOVSB. Обычное использование амперсанда (&) - для конкатенации, но в данном примере операнд <TAG> без амперсанда не будет работать. Если в макрокоманде не будет указан параметр B или W, то ассемблер сгенерирует предупреждающий комментарий и команду MOVSB (по умолчанию).

Примеры в кодовом сегменте трижды проверяют макрокоманду MOVIF: для параметра B, для параметра W и для неправильного параметра. Не следует делать попыток выполнения данной программы в том виде, как она приведена на рисунке, так как регистры CX и DX не обеспечены правильными значениями.

Предполагается, что рассматриваемая макрокоманда не является очень полезной и ее назначение здесь - проиллюстрировать условные директивы в простой форме. К данному моменту, однако, вы имеете достаточно информации для составления больших полезных макроопределений.

```
TITLE   MACRO7 (COM) Проверка директивы IFIDN
; ------------------------------------------
MOVIF   MACRO   TAG
        IFIDN   <&TAG>,<B>
        REP MOVSB
        EXITM
        ENDIF
        IFIDN   <&TAG>,<W>
        REP MOVSW
        ELSE
;       Не указан параметр B или W,
;           по умолчанию принято B
        REP MOVSB
        ENDIF
        ENDM
```

```
                    ; ------------------------------------------
0000                  CSIG    SEGMENT PARA 'Code'
                          ASSUME  CS:CSEG,DS:CSEG
                          ASSUME  SS:CSEG,ES:CSEG
0100                        ORG     100H
0100  EB 00              BEGIN:  JMP     SHORT MAIN
                    ;       ...
0102                  MAIN    PROC    NEAR
                          .LALL
                          MOVIF   B
                   +        IFIDN   <B>,<B>
0102  F3/A4              +        REP MOVSB
                   +        EXITM
                          MOVIF   W
                   +        ENDIF
                   +        IFIDN   <W>, <W>
0104  F3/A5              +        REP MOVSW
                   +        ENDIF
                          MOVIF
                   +        ENDIF
                   +        ELSE
                   + ;Не указан парам. B или W, по умолч.принято B
                   + ;------------------------------------------
0106  F3/A4              +        REP MOVSB
                   +        ENDIF
0108 C3                        RET
0109                  MAIN    ENDP
0109                  CSEG    ENDS
                          END     BEGIN
```

Рис.20.7. Использование директивы IFIDN

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- Макросредства возможны только для полной версии ассемблера (MASM).

- Использование макрокоманд в программах на ассемблере дает в результате более удобочитаемые программы и более производительный код.

- Макроопределение состоит из директивы MACRO, блока из одного или нескольких операторов, которые генерируются при макрорасширениях и директивы ENDM для завершения определения.

- Код, который генерируется в программе по макрокоманде, представляет собой макрорасширение.

- Директивы .SALL, .LALL и .XALL позволяют управлять распечаткой комментариев и генерируемого объектного кода в макрорасширении.

- Директива LOCAL позволяет использовать имена внутри макроопределений. Директива LOCAL кодируется непосредственно после директивы MACRO.

- Использование формальных параметров в макроопределении позволяет кодировать параметры, обеспечивающие большую гибкость макросредств.

- Библиотека макроопределений дает возможность использовать макрокоманды для различных ассемблерных программ.

- Условные директивы позволяют контролировать параметры макрокоманд.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

20.1. Напишите необходимые директивы: а) для подавления всех команд, которые генерирует макрокоманда и б) для распечатки только команд, генерирующих объектный код.

20.2. Закодируйте два макроопределения для умножения: а) MULTBY должна генерировать код для умножения байта на байт; б) MULTWD должна генерировать код для умножения слова на слово. Для множителя и множимого используйте в макроопределении формальные параметры. Проверьте выполнение макрокоманд на небольшой программе, в которой также определены необходимые области данных.

20.3. Запишите макроопределения из вопроса 20.2 в "макробиблиотеку". Исправьте программу для включения элементов библиотеки по директиве INCLUDE в первом проходе ассемблирования.

20.4. Напишите макроопределение BIPRINT, использующей BIOS INT 17H для печати. Макроопределение должно включать проверку состояния принтера и обеспечивать печать любых строк любой длины.

20.5. Измените макроопределение на рис.20.6 для проверки делителя на ноль (для обхода деления).

ГЛАВА 21 Компоновка программ

Ц е л ь: Раскрыть технологию программирования, включающую компоновку и выполнение ассемблерных программ.

ВВЕДЕНИЕ

Примеры программ в предыдущих главах состояли из одного шага ассемблирования. Возможно, однако, выполнение программного модуля, состоящего из нескольких ассемблированных программ. В этом случае программу можно рассматривать, как состоящую из основной программы и одной или более подпрограмм. Причины такой организации программ состоят в следующем:

- бывает необходимо скомпоновать программы, написанные на разных языках, например, для объединения мощности языка высокого уровня и эффективности ассемблера;
- программа, написанная в виде одного модуля, может оказаться слишком большой для ассемблирования;
- отдельные части программы могут быть написаны разными группами программистов, ассемблирующих свои модули раздельно;
- ввиду возможно большого размера выполняемого модуля, может появиться необходимость перекрытия частей программы в процессе выполнения.

Каждая программа ассемблируется отдельно и генерирует собственный уникальный объектный (OBJ) модуль. Программа компоновщик (LINK) затем

компонует объектные модули в один объединенный выполняемый (EXE) модуль. Обычно выполнение начинается с основной программы, которая вызывает одну или более подпрограмм. Подпрограммы, в свою очередь, могут вызывать другие подпрограммы.

На рис.21.1 показаны два примера иерархической структуры основной подпрограммы и трех подпрограмм. На рис.21.1.(а) основная программы вызывает подпрограммы 1, 2 и 3. На рис.21.1.(б) основная программа вызывает подпрограммы 1 и 2, а подпрограмма 1 вызывает подпрограмму 3.

Существует много разновидностей организации подпрограмм, но любая организация должна быть "понятна" и ассемблеру, и компоновщику, и этапу выполнения. Следует быть внимательным к ситуациям, когда, например, под программа 1 вызывает подпрограмму 2, которая вызывает подпрограмму 3 и, которая в свою очередь вызывает подпрограмму 1. Такой процесс, известный как рекурсия, может использоваться на практике, но при неаккуратном обращении может вызвать любопытные ошибки при выполнении.

```
а)      ------------¬           б) ------------¬
        ¦ Основная  ¦              ¦ Основная  ¦
        ¦ программа ¦              ¦ программа ¦
        L-----T------              L-----T------
    ----------+---------¬              -----+----¬
--------¬ --------¬ --------¬     --------¬ --------¬
¦ П/П 1 ¦ ¦ П/П 2 ¦ ¦ П/П 3 ¦     ¦ П/П 1 ¦ ¦ П/П 2 ¦
L-------- L-------- L--------     L---T---- L--------
                                --------¬
                                ¦ П/П 3 ¦
                                L--------
```

Рис.21.1. Иерархия программ.

МЕЖСЕГМЕНТНЫЕ ВЫЗОВЫ

Команды CALL в предыдущих главах использовались для внутрисегментных вызовов, т.е. для вызовов внутри одного сегмента. Внутрисегментный CALL может быть короткий (в пределах от +127 до -128 байт) или длинный (превышающий указанные границы). В результате такой операции "старое" значение в регистре IP запоминается в стеке, а "новый" адрес перехода загружается в этот регистр.

Например, внутрисегментный CALL может иметь следующий объектный код: E82000. Шест.E8 представляет собой код операции, которая заносит 2000 в виде относительного адреса 0020 в регистр IP. Затем процессор объединяет текущий адрес в регистре CS и относительный адрес в регистре IP для получения адреса следующей выполняемой команды. При возврате из процедуры команда RET восстанавливает из стека старое значение в регистре IP и передает управление таким образом на следующую после CALL команду.

Вызов в другой кодовый сегмент представляет собой межсегментный (длинный) вызов. Данная операция сначала записывает в стек содержимое регистра CS и заносит в этот регистр адрес другого сегмента, затем записывает в стек значение регистра IP и заносит новый относительный адрес в этот регистр.

Таким образом в стеке запоминаются и адрес кодового сегмента и смещение для последующего возврата из подпрограммы.

Например, межсегментный CALL может состоять из следующего объектного кода:

    9A 0002 AF04

Шест.9A представляет собой код команды межсегментного вызова которая записывает значение 0002 в виде 0200 в регистр IP, а значение AF04 в виде 04AF в регистр CS. Комбинация этих адресов указывает на первую выполняемую команду в вызываемой подпрограмме:

```
Кодовый сегмент          04AF0
Смещение в IP            0200
Действительный адрес     04CF0
```

При выходе из вызванной процедуры межсегментная команда возврата REP восстанавливает оба адреса в регистрах CS и IP и таким образом передает управление на следующую после CALL команду.

АТРИБУТЫ EXTRN и PUBLIC

Рассмотрим основную программу (MAINPROG), которая вызывает подпрограмму (SUBPROG) с помощью межсегментного CALL, как показано на рис.21.2.

Команда CALL в MAINPROG должна "знать", что SUBPROG существует вне данного сегмента (иначе ассемблер выдаст сообщение о том, что идентификатор SUBPROG не определен). С помощью директивы EXTRN можно указать ассемблеру, что ссылка на SUBPROG имеет атрибут FAR, т.е.определена в другом ассемблерном модуле. Так как сам ассемблер не имеет возможности точно определить такие ссылки, он генерирует "пустой" объектный код для последующего заполнения его при компановке:

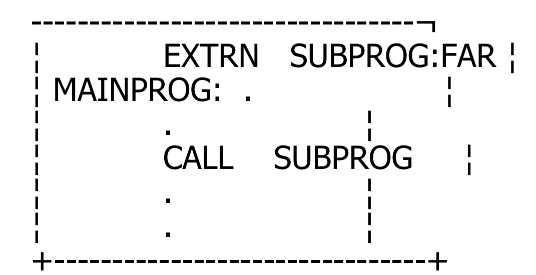
```
9A 0000 ---- E
```

Подпрограмма SUBPROG содержит директиву PUBLIC, которая указывает ассемблеру и компоновщику, что другой модуль должен "знать" адрес SUBPROG. В последнем шаге, когда оба модуля MAINPROG и SUBPROG будут успешно ассемблированы в объектные модули, они могут быть скомпонованы следующим образом:

```
Запрос компоновщика LINK:     Ответ:

Object Modules [.OBJ]:      B:MAINPROG+B:SUBPROG
Run File [filespec.EXE]:    B:COMBPROG (или другое имя)
List File [NUL.MAP]:        CON
Libraries [.LIB]:           [return]
```

Компоновщик устанавливает соответствия между адресами EXTRN в одном объектном модуле с адресами PUBLIC в другом и заносит необходимые относительные адреса. Затем он объединяет два объектных модуля в один выполняемый. При невозможности разрешить ссылки компоновщик выдает сообщения об ошибках. Следите за этими сообщениями прежде чем пытаться выполнить программу.



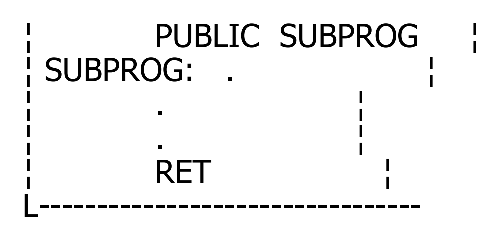


Рис.21.2. Межсегментный вызов.

Директива EXTRN

Директива EXTRN имеет следующий формат:

```
EXTRN     имя:тип [, ... ]
```

Можно определить более одного имени (до конца строки) или закодировать дополнительные директивы EXTRN. В другом ассемблерном модуле соответствующее имя должно быть определено и идентифицировано как PUBLIC. Тип элемента может быть ABS, BYTE, DWORD, FAR, NEAR, WORD. Имя может быть определено через EQU и должно удовлетворять реальному определению имени.

Директива PUBLIC

Директива PUBLIC указывает ассемблеру и компоновщику, что адрес указанного идентификатора доступен из других программ. Директива имеет следующий формат:

```
PUBLIC    идентификатор [, ... ]
```

Можно определить более одного идентификатора (до конца строки) или закодировать дополнительные директивы PUBLIC. Идентификаторы могут быть метками (включая PROC-метки), переменными или числами. Неправильными идентификаторами являются имена регистров и EQU-идентификаторы, определяющие значения более двух байт.

Рассмотрим три различных способа компановки программ.

ПРОГРАММА: ИСПОЛЬЗОВАНИЕ ДИРЕКТИВ EXTRN и PUBLIC ДЛЯ МЕТОК

```
                        page    60,132
                  TITLE   CALLMULL1 (EXE) Вызов подпрограммы умнож.
                        EXTRN   SUBMUL:FAR
                  ;-------------------------------------------
0000                STACKSG        SEGMENT PARA STACK 'Stack'
0000   40 [ ???? ]          DW      64 DUP(?)
0080                STACKSG        ENDS
                  ;-------------------------------------------
0000                DATASG  SEGMENT PARA 'Data'
0000  0140            QTY     DW      0140H
0002  2500            PRICE   DW      2500H
0004                DATASG  ENDS
                  ;-------------------------------------------
0000                CODESG  SEGMENT PARA 'Code'
0000                BEGIN   PROC    FAR
                        ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
0000  1E                    PUSH    DS
```

```
0001  2B C0                SUB     AX,AX
0003  50                  PUSH    AX
0004  B8 ---- R            MOV     AX,DATASG
0007  8E D8                MOV     DS,AX
0009  A1 0002 R             MOV     AX,PRICE    ;Загрузить стоимость
000C  8B 1E 0000 R           MOV     BX,QTY      ; и количество
0010  9A 0000 ---- E         CALL    SUBMUL      ;Вызвать подпрограмму
0015  CB                  RET
0016              BEGIN   ENDP
0016              CODESG  ENDS
                     END     BEGIN

Segments and Groups:
            N a m e        Size    Align   Combine Class
CODESG . . . . . . . . . . . .  0016    PARA    NONE    'CODE'
DATASG . . . . . . . . . . . .  0004    PARA    NONE    'DATA'
STACKSG. . . . . . . . . . . .  0080    PARA    STACK   'STACK'

Symbols:
            N a m e        Type    Value   Attr
BEGIN. . . . . . . . . . . . .  F PROC  0000    CODESG  Length=0016
PRICE. . . . . . . . . . . . .  L WORD  0002    DATASG
QTY. . . . . . . . . . . . . .  L WORD  0000    DATASG
SUBMUL . . . . . . . . . . . .  L FAR   0000            External


                     page    60,132
               TITLE   SUBMUL  Подпрограмма для умножения
               ;---------------------------------------------
0000              CODESG  SEGMENT PARA 'Code'
0000              SUBMUL  PROC    FAR
                     ASSUME  CS:CODESG
                     PUBLIC  SUBMUL
0000  F7 E3                MUL     BX    ;AX-стоимость, BX-количество
0002  CB                  RET          ;Произведение в DX:AX
0003              SUBMUL  ENDP
0003              CODESG  ENDS
                     END     SUBMUL

Segments and groups:
            N a m e        Size    Align   Combine Class
CODESG . . . . . . . . . . . .  0003    PARA    NONE    'CODE'

Symbols:
            N a m e        Type    Value   Attr
SUBMUL . . . . . . . . . . . .  F PROC  0000    CODESG  Clobal Length=0003
```

```
LINK
IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp 1981, 1985
Object Modules: B:CALLMUL1+B:SUBMUL1
Run File: [B:CALLMUL1.EXE]: <return>
List File:[NUL.MAP]: CON
Libraries [.LIB]: <return>

 Start   Stop  Length  Name           Class

 00000H 00015H 0016H  CODESG          CODE  <--Примечание: 2 кодовых
 00020H 00022H 0003H  CODESG          CODE  <--  сегмента
 00030H 00033H 0004H  DATASG          DATA
```

```
 00040H 000BFH 0080H  STACKSG          STACK

Program entry point at 0000:0000
```

Рис. 21.3. Использование директив EXTRN и PUBLIC.

Программа на рис.21.3 состоит из основной программы CALLMUL1 и подпрограммы SUBMUL1. В основной программе определены сегменты для стека, данных и кода. В сегменте данных определены поля QTY и PRICE. В кодовом сегменте регистр AX загружается значением PRICE, а регистр BX - значением QTY, после чего происходит вызов подпрограммы. Директива EXTRN в основной программе определяет SUBMUL как точку входа в подпрограмму.

Подпрограмма содержит директиву PUBLIC (после ASSUME), которая указывает компоновщику, что точкой входа для выполнения является метка SUBMUL. Подпрограмма выполняет умножение содержимого регистра AX (цена) на содержимое регистра BX (количество). Результат умножения вырабатывается в регистровой паре DX:AX в виде шест.002E 4000.

Так как подпрограмма не определяет каких-либо данных, то ей не требуется сегмент данных. Если бы подпрограмма имела сегмент данных, то только она одна использовала бы свои данные.

Также в подпрограмме не определен стековый сегмент, так как она использует те же стековые адреса, что и основная программа. Таким образом, стек определенный в основной программе является доступным и в подпрограмме. Для компоновщика необходимо обнаружить по крайней мере один стек и определение стека в основной программе является достаточным.

Рассмотрим теперь таблицы идентификаторов, вырабатываемые после каждого ассемблирования. Обратите внимание, что SUBMUL в таблице идентификаторов для основной программы имеет атрибуты FAR и External (внешний), а для подпрограммы - F (для FAR) и Global (глобальный). Этот последний атрибут указывает, что данное имя доступно из вне подпрограммы, т.е. глобально.

Карта компановки (в конце листинга) отражает организацию программы в памяти. Заметьте, что здесь имеются два кодовых сегмента (для каждого ассемблирования) с разными стартовыми адресами. Последовательность расположения кодовых сегментов соответствует последовательности указанных для компановки объектных модулей (обычно основная программа указывается первой). Таким образом, относительный адрес начала основной программы - шест.00000, а подпрограммы - шест.00020.

При трассировке выполнения программы можно обнаружить, что команда CALL SUBMUL имеет объектный код

```
9A 0000 D413
```

Машинный код для межсегментного CALL - шест.9A. Эта команда сохраняет в стеке регистр IP и загружает в него значение 0000, сохраняет в стеке значение шест.13D2 из регистра CS и загружает в него шест.D413. Следующая выполняемая команда находится по адресу в регистровой паре CS:IP т.е. 13D40 плюс 0000. Обратите внимание, что основная программа начинается по адресу в регистре CS, содержащему шест.13D2, т.е. адрес 13D20. Из карты компановки видно, что подпрограмма начинается по относительному адресу шест.0020. Складывая эти два значения, получим действительный адрес кодового сегмента для подпрограммы:

```
Адрес в CS            13D20
Смещение в IP          0020
Действительный адрес   13D40
```

Компоновщик определяет это значение точно таким же образом, и подставляет его в операнд команды CALL.

ПРОГРАММА: ИСПОЛЬЗОВАНИЕ ДИРЕКТИВЫ PUBLIC В КОДОВОМ СЕГМЕНТЕ

```
                      page    60,132
                 TITLE   CALLMUL2 (EXE)  Вызов подпрограммы умнож.
                      EXTERN  SUBMUL:FAR
                 ;-------------------------------------------
0000               STACKSG        SEGMENT PARA STACK 'Stack'
0000  40 [????]                 DW      64 DUP(?)
0080               STACKSG        ENDS
                 ;-------------------------------------------
0000               DATASG  SEGMENT PARA 'Data'
0000     0140         QTY     DW      0140H
0002    2500          PRICE   DW      2500H
0004               DATASG  ENDS
                 ;-------------------------------------------
0000               CODESG  SEGMENT PARA  PUBLIC 'Code'
0000               BEGIN   PROC    FAR
                      ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
0000    1E                 PUSH    DS
0001    2B C0               SUB     AX,AX
0003    50                 PUSH    AX
0004    B8 ---- R           MOV     AX,DATASG
0007    8E D8               MOV     DS,AX
0009    A1 0002 R            MOV     AX,PRICE   ;Загрузить стоимость
000C    8B 1E 0000 R          MOV     BX,QTY     ; и количество
0010    9A 0000 ---- E       CALL    SUBMUL     ;Вызвать подпрограмму
0015    CB                 RET
0016               BEGIN   ENDP

0016               CODESG  ENDS
                      END     BEGIN
```

```
Segments and Group:
            N a m e        Size    Align   Combine Class
CODESG . . . . . . . . . . . . .0016    PARA    PUBLIC  'CODE'
DATASG . . . . . . . . . . . . .0004    PARA    NONE    'DATA'
STACKSG. . . . . . . . . . . . .0080    PARA    STACK   'STACK'

Symbols:
            N a m e        Type    Value   Attr
BEGIN. . . . . . . . . . . . . F PROC   0000    CODESG Lenght=0016
PRICE. . . . . . . . . . . . . L WORD   0002    DATASG
QTY. . . . . . . . . . . . . . L WORD   0000    DATASG
SUBMUL . . . . . . . . . . . . L FAR    0000          External
```

```
                      page    60,132
                 TITLE   SUBMUL2 Вызываемая подпрограмма умножения
                 ;-------------------------------------------
0000               CODESG  SEGMENT PARA    PUBLIC 'CODE'
0000               SUBMUL  PROC    FAR
                      ASSUME  CS:CODESG
                      PUBLIC  SUBMUL
0000    F7 E3               MUL     BX      ;AX-стоимость, BX-количество
0002    CB                 RET            ;Произведение в DX:AX
```

```
0003                SUBMUL  ENDP
0003                CODESG  ENDS
                        END    SUBMUL

Segments and Groups:
            N a m e         Size    Align   Combine Class
CODESG. . . . . . . . . . . . . 0003    PARA    PUBLIC  'CODE'

Symbols:
            N a m e         Type    Value   Attr
SUBMUL. . . . . . . . . . . . .F PROC   0000    CODESG  Global  Length=0003


LINK
IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp 1981, 1985
Object Modules: B:CALLMUL2+B:SUBMUL2
Run File: [B:CALLMUL2.EXE]: <return>
List File: [NUL.MAP]: CON
Libraries [.LIB]: <return>

 Start   Stop  Length  Name            Class

 00000H 00022H 0023H  CODESG           CODE  <-- Примечание: 1 сегмент кода
 00030H 00033H 0004H  DATASG           DATA
 00040H 000BFH 0080H  STACKSG          STACK

Program entry point at 0000:0000
```

Рис.21.4. Кодовый сегмент, определенный как PUBLIC.

Следующий пример на рис.21.4 представляет собой вариант программы на рис.21.3. Имеется одно изменение в основной программе и одно - в подпрограмме. В обоих случаях в директиве SEGMENT используется атрибут PUBLIC:

```
CODESG    SEGMENT   PARA PUBLIC 'CODE'
```

Рассмотрим результирующую карту компоновки и объектный код команды CALL.

Из таблицы идентификаторов (в конце каждого листинга ассемблирования) следует: обобщенный тип кодового сегмента CODESG - PUBLIC (на рис.21.3 было NONE). Но более интересным является то, что карта компановки в конце листинга показывает теперь только один кодовый сегмент! Тот факт, что оба сегмента имеют одни и те же имя (CODESG), класс ('CODE') и атрибут PUBLIC, заставил компоновщика объединить два логических кодовых сегмента в один физический кодовый сегмент. Кроме того, при трассировке выполнения программы можно обнаружить, что теперь команда вызова подпрограммы имеет следующий объектный код:

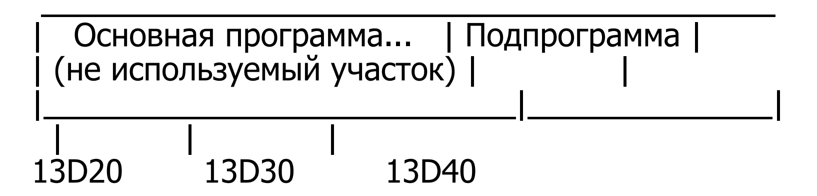
```
9A 2000 D213
```

Эта команда заносит шест.2000 в регистр IP и шест.D213 в регистр CS. Так как подпрограмма находится в общем с основной программой кодовом сегменте, то в регистре CS устанавливается тот же стартовый адрес - шест.D213. Но теперь смещение равно шест.0020:

```
Адрес в CS:          13D20
Смещение в IP:          0020
```

Действительный адрес: 13D40

Таким образом, кодовый сегмент подпрограммы начинается, очевидно, по адресу шест.13D40. Правильно ли это? Карта компановки не дает ответа на этот вопрос, но можно определить адрес по листингу основной программы, которая заканчивается на смещении шест.0016. Так как кодовый сегмент для подпрограммы определен как SEGMENT, то он должен начинаться на границе параграфа, т.е. его адрес должен нацело делиться на шест.10 или правая цифра адреса должна быть равна 0. Компоновщик размещает подпрограмму на ближайшей границе параграфа непосредственно после основной программы - этот относительный адрес равен шест.00020. Поэтому кодовый сегмент подпрограммы начинается по адресу 13D20 плюс 0020 или 13D40.



Рассмотрим, каким образом компоновщик согласует данные, определенные в основной программе и имеющие ссылки из подпрограммы.

ПРОГРАММА: ОБЩИЕ ДАННЫЕ В ПОДПРОГРАММЕ

```
                        page    60,132
                   TITLE   CALLMUL3 (EXE)  Вызов подпрограммы
                   ;                    для умножения
                        EXTRN   SUBMUL:FAR
                        PUBLIC  QTY,PRICE
                   ;-----------------------------------------------
0000                 STACKSG        SEGMENT PARA STACK 'Stack'
0000   40 [????]             DW      64 DUP(?)
0080                 STACKSD        ENDS
                   ;-----------------------------------------------
0000                 DATASG  SEGMENT PARA PUBLIC 'Data'
0000   0140            QTY     DW      0140H
0002   2500            PRICE   DW      2500H
0004                 DATASG  ENDS
                   ;-----------------------------------------------
0000                 CODESG  SEGMENT PARA PUBLIC 'Code'
0000                 BEGIN   PROC    FAR
                        ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
0000   1E                   PUSH    DS
0001   2B C0                 SUB     AX,AX
0003   50                   PUSH    AX
0004   B8 ---- R             MOV     AX,DATASG
0007   8E D8                 MOV     DS,AX
0009   9A 0000 ---- E          CALL    SUBMUL    ;Вызвать подпрограмму
000E   CB                    RET
000F                 BEGIN   ENDP

000F                 CODESG  ENDS
                        END     BEGIN
```

Segments and Groups:

```
             N a m e         Size    Align   Combine Class
CODESG . . . . . . . . . . . .  000F    PARA    PUBLIC  'CODE'
DATASG . . . . . . . . . . . .  0004    PARA    PUBLIC  'DATA'
STACKSG. . . . . . . . . . . .  0080    PARA    STACK   'STACK'

Symbols:
             N a m e         Type    Value   Attr
BEGIN. . . . . . . . . . . . .  F PROC  0000    CODESG  Length=000F
PRICE. . . . . . . . . . . . .  L WORD  0002    DATASG  Global
QTY. . . . . . . . . . . . . .  L WORD  0000    DATASG  Global
SUBMUL . . . . . . . . . . . .  L FAR   0000            External


                       page    60,132
                 TITLE   SUBMUL  Подпрограмма для умножения
                       EXTRN   QTY:WORD,PRICE:WORD
                 ;-----------------------------------------------
0000                CODESG  SEGMENT PARA PUBLIC 'CODE'
0000                SUBMUL  PROC    FAR
                       ASSUME  CS:CODESG
                       PUBLIC  SUBMUL
0000    A1 0000 E              MOV     AX,PRICE
0003    8B 1E 0000 E           MOV     BX,QTY
0007    F7 E3                  MUL     BX        ;Произведение в DX:AX
0009    CB                     RET
000A                SUBMUL  ENDP
000A                CODESG  ENDS
                       END     SUBMUL
```

```
Segments and Groups:
             N a m e         Size    Align  Combine Class
CODESG . . . . . . . . . . . .  000A    PARA   PUBLIC  'CODE'

Symbols:
             N a m e         Type    Value  Attr
PRICE. . . . . . . . . . . . .  V WORD  0000          External
QTY. . . . . . . . . . . . . .  V WORD  0000          External
SUBMUL . . . . . . . . . . . .  F PROC  0000   CODESG  Global  Length=000A
```

```
LINK
IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp 1981, 1985
Object Modules: B:CALLMUL3+B:SUBMUL3
Run File: [B:CALLMUL3.EXE]: <return>
List File: [NUL.MAP]: CON
Libraries [.LIB]: <return>

 Start   Stop  Length  Name            Class

 00000H 00019H 001AH  CODESG           CODE
 00030H 00033H 0004H  DATASG           DATA
 00040H 000BFH 0080H  STACKSG          STACK

PROGRAM entry point at 0000:0000
```

Рис.21.5. Общие данные в подпрограмме.

Наличие общих данных предполагает возможность обработки в одном ассемблерном модуле данных, которые определены в другом ассемблерном модуле. Изменим предыдущий пример так, чтобы области QTY и PRICE по-прежнему определялись в основной программе, но загрузка значений из этих областей в регистры BX и AX выполнялась в подпрограмме. Такая программа приведена на рис.21.5. В ней сделаны следующие изменения:

- В основной программе имена QTY и PRICE определены как PUBLIC. Сегмент данных также определен с атрибутом PUBLIC. Обратите внимание на атрибут Global (глобальный) для QTY и PRICE в таблице идентификаторов.

- В подпрограмме имена QTY и PRICE определены как EXTRN и WORD. Такое определение указывает ассемблеру на длину этих полей в 2 байта. Теперь ассемблер сгенерирует правильный код операции для команд MOV, а компоновщик установит значения операндов. Заметьте, что имена QTY и PRICE в таблице идентификаторов имеют атрибут External (внешний).

Команды MOV в листинге подпрограммы имеют следующий вид:

```
A1 0000 E      MOV  AX,PRICE
8B 1E 0000 E   MOV  BX,QTY
```

В объектном коде шест.A1 обозначает пересылку слова из памяти в регистр AX, а шест.8B - пересылку слова из памяти в регистр BX (объектный код для операций с регистром AX чаще требует меньшее число байтов, чем с другими регистрами). Трассировка выполнения программы показывает, что компоновщик установил в объектном коде следующие операнды:

```
A1 0200
8B 1E 0000
```

Объектный код теперь идентичен коду сгенерированному в предыдущем примере, где команды MOV находились в вызывающей программе. Это логичный результат, так как операнды во всех трех программах базировались по регистру DS и имели одинаковые относительные адреса.

Основная программа и подпрограмма могут определять любые другие элементы данных, но общими являются лишь имеющие атрибуты PUBLIC и EXTRN.

Следуя основным правилам, рассмотренным в данной главе, можно теперь компоновать программы, состоящие более чем из двух ассемблерных модулей и обеспечивать доступ к общим данным из всех модулей. При этом следует предусматривать стек достаточных размеров - в разумных пределах, для больших программ определение 64 слов для стека бывает достаточным.

В гл.23 будет рассмотрены дополнительные свойства сегментов, включая определение более одного сегмента данных и кодового сегмента в одном ассемблерном модуле и использование директивы GROUP для объединения сегментов в один общий сегмент.

ПЕРЕДАЧА ПАРАМЕТРОВ

```
                          page    60,132
                    TITLE   CALLMULL4 (EXE) Передача параметров
                    ;                    в подпрограмму
                          EXTRN   SUBMUL:FAR
                    ;-----------------------------------------------
0000                   STACKSG        SEGMENT PARA STACK 'Stack'
0000    40 [ ???? ]             DW      64 DUP(?)
```

```
0080                 STACKSG         ENDS
                    ;-----------------------------------------------
0000                 DATASG  SEGMENT PARA 'Data'
0000  0140              QTY     DW      0140H
0002  2500              PRICE   DW      2500H
0004                 DATASG  ENDS
                    ;-----------------------------------------------
0000                 CODESG  SEGMENT PARA PUBLIC 'Code'
0000                 BEGIN   PROC    FAR
                          ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
0000  1E                       PUSH    DS
0001  2B C0                     SUB     AX,AX
0003  50                       PUSH    AX
0004  B8 ---- R                 MOV     A,DATASG
0007  8E D8                     MOV     DS,AX
0009  FF 36 0002 R                 PUSH    PRICE
000D  FF 36 0000 R                 PUSH    QTY
0011  9A 0000 ---- E               CALL    SUBMUL      ;Вызвать подпрограмму
0016  CB                       RET
0017                 BEGIN   ENDP
0017                 CODESG  ENDS
                          END     BEGIN
```

```
Segments and Groups:
             N a m e        Sise    Align   Combine Class
CODESG . . . . . . . . . . . .  0017    PARA    NONE    'CODE'
DATASG . . . . . . . . . . . .  0004    PARA    NONE    'DATA'
STACKSG. . . . . . . . . . . .  0080    PARA    STACK   'STACK'

Symbols:
             N a m e        Type    Value   Attr
BEGIN. . . . . . . . . . . . .  F PROC  0000    CODESG  Length=0017
PRICE. . . . . . . . . . . . .  L WORD  0002    DATASG
QTY. . . . . . . . . . . . . .  L WORD  0000    DATASG
SUBMUL . . . . . . . . . . . .  L FAR   0000            External
```

```
                          page    60,132
                    TITLE   SUBMUL  Вызываемая подпрограмма умножения
0000                 CODESG  SEGMENT PARA PUBLIC 'Code'
0000                 SUBMUL  PROC    FAR
                          ASSUME  CS:CODESG
                          PUBLIC  SUMBUL
0000    55                     PUSH    BP
0001    8P EC                   MOV     BP,SP
0003    8B 46 08                 MOV     AX,[BP+8]    ;Стоимость
0006    8B 5E 06                 MOV     BX,[BP+6]    ;Количество
0009    F7 E3                   MUL     BX          ;Произведение в DX:AX
000B    5D                      POP     BP
000F                 SUMBUL  ENDP
000F                 CODESG  ENDS
                          END
```

```
Segments and Groups:
             N a m e        Size    Align   Combine Class
CODESG . . . . . . . . . . . .  000F    PARA    PUBLIC  'CODE'

Symbols:
             N a m e        Type    Value   Attr
```

```
SUBMUL . . . . . . . . . . . .  F PROC  0000    CODESG  Global Length=000F
```

```
LINK
IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp 1981, 1985
Object Modules: B:CALLMUL4+B:SUBMUL4
Run File: [B:CALLMUL4.EXE]: <return>
List File: [NUL.MAP]: CON
Libraries [.LIB]: <return>

 Start   Stop  Length  Name            Class

 00000H 00019H 001AH  CODESG           CODE
 00030H 00033H 0004H  DATASG           DATA
 00040H 000BFH 0080H  STACKSG          STACK

PROGRAM entry point at 0000:0000
```

Рис.21.6. Передача параметров.

Другим способом обеспечения доступа к данным из вызываемой подпрограммы является передача параметров. В этом случае вызывающая программа физически передает данные через стек. Каждая команда PUSH должна записывать в стек данные размером в одно слово из памяти или из регистра.

Программа, приведенная на рис.21.6, прежде чем вызвать подпрограмму SUBMUL заносит в стек значения из полей PRICE и QTY. После команды CALL стек выглядит следующим образом:

```
... | 1600 | D213 | 4001 | 0025 | 0000 | C213 |
       6      5      4      3      2      1
```

1. Инициализирующая команда PUSH DS заносит адрес сегмента в стек. Этот адрес может отличаться в разных версиях DOS.
2. Команда PUSH AX заносит в стек нулевой адрес.
3. Команда PUSH PRICE заносит в стек слово (2500).
4. Команда PUSH QTY заносит в стек слово (0140).
5. Команда CALL заносит в стек содержимое регистра CS (D213)
6. Так как команда CALL представляет здесь межсегментный вызов,то в стек заносится также содержимое регистра IP (1600).

Вызываемая программа использует регистр BP для доступа к параметрам в стеке, но прежде она запоминает содержимое регистра BP, записывая его в стек. В данном случае, предположим, что регистр BP содержит нуль, тогда нулевое слово будет записано в вершине стека (слева).

Затем программа помещает в регистр BP содержимое из регистра SP, так как в качестве индексного регистра может использоваться регистр BP, но не SP. Команда загружает в регистр BP значение 0072. Первоначально регистр SP содержал размер пустого стека, т.е. шест.80. Запись каждого слова в стек уменьшает содержимое SP на 2:

```
 | 0000 | 1600 | D213 | 4001 | 0025 | 0000 | C213 |
   |      |      |      |      |      |      |
SP: 72     74     76     78     7A     7C     7E
```

Так как BP теперь также содержит 0072, то параметр цены (PRICE) будет по адресу BP+8, а параметр количества (QTY) - по адресу BP+6. Программа пересылает эти величины из стека в регистры AX и BX соответственно и выполняет умножение.

Перед возвратом в вызывающую программу в регистре BP восстанавливается первоначальное значение, а содержимое в регистре SP увеличивается на 2, с 72 до 74.

Последняя команда RET представляет собой "длинный" возврат в вызывающую программу. По этой команде выполняются следующие действия:

- Из вершины стека восстанавливается значение регистра IP (1600)
- Содержимое регистра SP увеличивается на 2, от 74 до 76.
- Из новой вершины стека восстанавливается значение регистра CS (D213).
- Содержимое регистра SP увеличивается на 2 от 76 до 78.

Таким образом осуществляется корректный возврат в вызывающую программу. Осталось одно небольшое пояснение. Команда RET закодирована как

```
RET  4
```

Параметр 4 представляет собой число байт в стеке использованных при передаче параметров (два слова в данном случае). Команда RET прибавит этот параметр к содержимому регистра SP, получив значение 7C. Таким образом, из стека исключаются ненужные больше параметры. Будьте особенно внимательны при восстановлении регистра SP - ошибки могут привести к непредсказуемым результатам.

КОМПОНОВКА ПРОГРАММ НА BASIC-ИНТЕРПРЕТАТОРЕ И АССЕМБЛЕРЕ

В руководстве по языку BASIC для IBM PC приводятся различные методы связи BASIC-интерпретатора и программ на ассемблере. Для этого имеются две причины: сделать возможным использование BIOS-прерываний через ассемблерные модули и создать более эффективные программы. Цель данного раздела - дать общий обзор по данному вопросу; повторять здесь технические подробности из руководства по языку BASIC нет необходимости.

Для связи с BASIC ассемблерные программы кодируются, транслируются и компонуются отдельно. Выделение памяти для подпрограмм на машинном языке может быть либо внутри, либо вне 64 Кбайтовой области памяти, которой ограничен BASIC. Выбор лежит на программисте.

Существует два способа загрузки машинного кода в память: использование оператора языка BASIC - POKE или объединение скомпонованного модуля с BASIC-программой.

Использование BASIC-оператора POKE

Хотя это и самый простой способ, но он удобен только для очень коротких подпрограмм. Способ заключается в том, что сначала определяется объектный код ассемблерной программы по LST-файлу или с помощью отладчика DEBUG. Затем шестнадцатиричные значения кодируются непосредственно в BASIC-программе в операторах DATA. После этого с помощью BASIC-оператора READ считывается каждый байт и оператором POKE заносится в память для выполнения.

Компановка ассемблерных модулей

С большими ассемблерными подпрограммами обычно проще иметь дело, если они оттранслированы и скомпонованы как выполнимые (EXE) модули. Необходимо организовать BASIC-программу и выполнимый модуль в рабочую программу. При работе с BASIC-программой не забывайте пользоваться командой BSAVE (BASIC save) для сохранения программы и BLOAD - для загрузки ее перед

выполнением.

Прежде чем кодировать BASIC- и ассемблерную программы, необходимо решить, каким из двух способов они будут связаны. В языке BASIC возможны два способа: функция USR и оператор CALL. В обоих способах регистры DS, ES и SS на входе содержат указатель на адресное пространство среды BASIC. Регистр CS содержит текущее значение, определенное последним оператором DEF SEG (если он имеется). Стековый указатель SP указывает на стек, состоящий только из восьми слов, так что может потребоваться установка другого стеке в подпрограмме. В последнем случае необходимо на входе сохранить значение указателя текущего стека, а при выходе восстановить его. В обоих случаях при выходе необходимо восстановить значение сегментных регистров и SP и обеспечить возврат в BASIC с помощью межсегментного возврата RET.

Скомпонуйте ваш ассемблированный объектный файл так, что бы он находился в старших адресах памяти. Для этого используется параметр HIGH при ответе на второй запрос компоновщика, например, B:имя/HIGH. Затем с помощью отладчика DEBUG необходимо загрузить EXE-подпрограмму и по команде R определить значения в регистрах CS и IP: они показывают на стартовый адрес подпрограммы. Находясь в отладчике укажите имя (команда N) BASIC и загрузите его командой L.

Два способа связи BASIC-программы и EXE-подпрограммы - использование операторов USR или CALL. Работая в отладчике, необходимо определить стартовый адрес EXE-подпрограммы и, затем, указать этот адрес или в операторе USRn или в CALL. В руководстве по языку BASIC для IBM PC детально представлено описание функции USRn и оператора CALL с различными примерами.

Программа: Компоновка BASIC и ассемблера

```
LOAD"D:BASTEST.BAS

LIST
010     CLEAR ,32768!
020     ' для BLOAD
030     ' для DEFSEG
040     ' для точки входа в CALL
050     ' для вызова ASM-модуля
060     FOR N = 1 TO 5
070     INPUT "Hours "; H
080     INPUT "Rate "; R
090     W = H * R
100     PRINT "Wage = " W
110     NEXT N
120     END
```

```
TITLE   LINKBAS Ассемблерная подпрограмма, вызываемая из BASIC
CODESG  SEGMENT PARA 'CODE'
        ASSUME  CS:CODESG
CLRSCRN PROC    FAR
        PUSH    BP              ;Сохранить BP
        MOV     BP,SP           ;База списка параметров
        MOV     AX,0600H        ;Функция прокрутки
        MOV     BH,07           ; всего
        MOV     CX,0000         ; экрана
        MOV     DX,184FH
        INT     10H
        POP     BP
```

```
       RET                   ;Завершить подпрограмму
CLRSCRN ENDP
CODESG  ENDS
       END
```

Рис.21.7. Основная программа на языке BASIC и подпрограмма на ассемблере.

Рассмотрим теперь простой пример компановки программы для BASIC-интерпретатора и подпрограммы на ассемблере. В этом примере BASIC-программа запрашивает ввод значений времени и расценки и выводит на экран их произведение - размер зарплаты. Цикл FOR-NEXT обеспечивает пятикратное выполнение ввода и затем программа завершается. Пусть BASIC-программа вызывает ассемблерный модуль, который очищает экран.

На рис. 21.7 приведена исходная BASIC-программа и ассемблерная подпрограмма. Обратите внимание на следующие особенности BASIC-программы: оператор 10 очищает 32К байт памяти; операторы 20, 30, 40 и 50 временно содержат комментарии.

Позже мы вставим BASIC-операторы для связи с ассемблерным модулем. BASIC-программу можно сразу проверить. Введите команду BASIC и затем наберите все пронумерованные операторы так, как они показаны в примере. Для выполнения программы нажмите F2. Не забудьте сохранить текст программы с помощью команды

```
SAVE "B:BASTEST.BAS"
```

Обратите внимание на следующие особенности ассемблерной подпрограммы:

- отсутствует определение стека, так как его обеспечивает BASIC; программа не предусмотрена для отдельного выполнения и не может быть выполнена;
- подпрограмма сохраняет в стеке содержимое регистра BP и записывает значение регистра SP в BP;
- подпрограмма выполняет очистку экрана, хотя она может быть изменена для выполнения других операций, таких как прокрутка экрана вверх или вниз или установка курсора.

Все что осталось - это связать эти программы вместе. Следующие действия предполагают, что системная дискета (DOS) находится на дисководе A, а рабочие программы - на дисководе B:

1. Наберите ассемблерную подпрограмму, сохраните ее под именем B:LINKBAS.ASM и оттранслируйте ее.
2. Используя компоновщик LINK, сгенерируйте объектный модуль, который будет загружаться в старшие адреса памяти:

   ```
   LINK B:LINKBAS,B:LINKBAS/HIGH,CON;
   ```

3. С помощью отладчика DEBUG загрузите BASIC - компилятор: DEBUG BASIC.COM.
4. По команде отладчика R выведите на экран содержимое регистров. Запишите значения в регистрах SS, CS и IP.
5. Теперь установите имя и загрузите скомпонованный ассемблерный модуль следующими командами:

   ```
   N B:LINKBAS.EXE
   L
   ```

6. По команде R выведите на экран содержимое регистров и запишите значения в CX, CS и IP.
7. Замените содержимое регистров SS, CS и IP значениями из шага 4. (Для

этого служат команды R SS, R CS и R IP).

8. Введите команду отладчика G (go) для передачи управления в BASIC. На экране должен появиться запрос из BASIC-программы.
9. Для того, чтобы сохранить ассемблерный модуль, введите следующие команды (без номеров операторов):

```
DEF SEG = &Hxxxx (значение в CS из шага 6)
BSAVE "B:CLRSCRN.MOD",0,&Hxx (значение в CX из шага 6)
```

Первая команда обеспечивает адрес загрузки модуля в память для выполнения. Вторая команда идентифицирует имя модуля, относительную точку входа и размер модуля. По второй команде система запишет модуль на дисковод B.

10. Теперь необходимо модифицировать BASIC-программу для компоновки. Можно загрузить ее сразу, находясь в отладчике, но вместо этого наберите команду SYSTEM для выхода из BASIC и, затем, введите Q для выхода из отладчика DEBUG. На экране должно появиться приглашение DOS
11. Введите команду BASIC, загрузите BASIC-программу и выведите ее на экран:

```
BASIC
LOAD "B:BASTEST.BAS"
LIST
```

12. Измените операторы 20, 30, 40 и 50 следующим образом:

```
20   BLOAD "B:CLRSCRN.MOD"
30   DEF SEG = &Hxxxx (значение в CS из шага 6)
40   CLRSCRN = 0      (точка входа в подпрограмму)
50   CALL CLRSCRN     (вызов подпрограммы)
```

13. Просмотрите, выполните и сохраните измененную BASIC-программу.

Если BASIC-программа и ассемблерные команды были введены правильно, а также правильно установлены шестнадцатеричные значения из регистров, то связанная программа должна сразу очистить экран и выдать запрос на ввод времени и расценки. На рис.21.8 приведен протокол всех шагов - но некоторые значения могут отличаться в зависимости от версии операционной системы и размера памяти.

Приведенный пример выбран намеренно простым только для демонстрации компоновки. Можно использовать более сложную технологию, используя передачу параметров из BASIC-программы в ассемблерную подпрограмму с помощью оператора

```
CALL подпрограмма (параметр-1,параметр-2,...)
```

Ассемблерная подпрограмма может получить доступ к этим параметрам, используя регистр BP в виде [BP], как это делалось ранее на рис.21.3. В этом случае необходимо определить операнд в команде RET, соответствующий длине адресов параметров в стеке. Например, если оператор CALL передает три параметра то возврат должен быть закодирован в виде RET 6.

```
D>LINK

IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp. 1981, 1985

Object Modules [.OBJ]: LINKBAS
Run File [LINKBAS.EXE]: LINKBAS/HIGH
```

```
List File [NUL.MAP]: CON
Libraries [.LIB]:
Warning: no stack segment

 Start  Stop   Length Name              Class
 00000H 00011H 00012H CODESG            CODE
D>DEBUG BASIC.COM
-R
AX=0000  BX=0000  CX=0012  DX=0000  SP=FFFF  BP=0000  SI=0000  DI=0000
DS=1410  ES=1410  SS=1410  CS=1410  IP=0100   NV UP EI PL NZ NA PO NC
1410:0100 E9E03E        JMP     3FE3
-N D:LINKBAS.EXE
-L
-R
AX=FFA3  BX=0000  CX=0012  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1410  ES=1410  SS=9FE0  CS=9FE0  IP=0000   NV UP EI PL NZ NA PO NC
9FE0:0000  55
-R SS
SS 9FE0
:1410
-R CS
CS 9FE0
:1410
-R IP
IP 0000
:0100
-G
Ok
DEF SEG = &H9EF0
Ok
BSAVE "D:CLRSCREEN.MOD",0,&H12
Ok
SYSTEM
Program terminated normally
-Q
D>BASIC
IBM Personal Computer Basic
Ver4sion D3.10 Copyright IBM Corp. 1981, 1985
61310 Bytes free
Ok
LOAD"D:BASTEST.BAS
Ok
20      BLOAD "D:CLRSCREEN.MOD"
30      DEF SEG = &H9FE0
40      CLRSCRN = 0
50      CALL CLRSCRN
LIST

10      CLEAR ,32768!
20      BLOAD "D:CLRSCRN.MOD"
30      DEF SEG = &H9FE0
40      CLRSCRN = 0
50      CALL CLRSCRN
60      FOR N = 1 TO 5
70      INPUT "HOURS"; H
80      INPUT "rATE"; R
90      W = H * R
100      PRINT "WAGE = " W
110      NEXT N
120      END
Ok
```

Рис.21.8. Этапы связи BASIC и ассемблера.

КОМПОНОВКА ПРОГРАММ НА ЯЗЫКЕ PASCAL И АССЕМБЛЕРЕ

```
program pascall ( input, output );

   procedure move_cursor( const row: integer;
                    const col: integer ); extern;
   var
      temp_row:      integer;
      temp_col:      integer;

   begin
      write( 'Enter cursor row: ' );
      readln( temp_row );

      write( 'Enter cursor column:' );
      readln( temp_col );

      move_cursor( temprow, temp_col );
      write( 'New cursor location' );
    end.
```

```
TITLE   MOVCUR  Подпрограмма на ассемблере,
;              вызываемая из программы на Паскале
      PUBLIC  MOVE_CURSOR
;-------------------------------------------------------
;   MOVE_CURSOR:      Устанавливает курсор
;                     по переданным параметрам
;   Параметры: const row      Строка и столбец
;              const col      для установки курсора
;   Возвращаемое значение:    Отсутствует
;-------------------------------------------------------
CODESEG SEGMENT PARA PUBLIC 'CODE'

MOVE_CURSOR PROC FAR
      ASSUME  CS:CODESEG
ROWWPAR EQU     8             ;Параметр "строка"
COLPAR  EQU     6             ;Параметр "столбец"

      PUSH    BP              ;Сохранить регистр BP
      MOV     BP,SP           ;Установить BP на параметры

      MOV     SI,[BP+ROWPAR]  ;SI указывает на строку
      MOV     DH,[SI]         ;Поместить столбец в DL

      MOV     AH,02           ;Функция установки курсора
      SUB     BH,BH           ;Страница #0
      INT     10H

      POP     BP              ;Вернуться
      RET     4               ; в вызывающую программу
MOVE_CURSOR      ENDP
CODESEG ENDS
```

```
    END
```

Рис.21.9. Компановка PASCAL-ассемблер.

В данном разделе показано, как можно установить связь между программами на языке PASCAL фирм IBM и MicroSoft с программами на ассемблере. На рис.21.9 приведен пример связи простой PASCAL-программы с ассемблерной подпрограммой. PASCAL-программа скомпилирована для получения OBJ-модуля, а ассемблерная программа оттранслирована также для получения OBJ-модуля. Программа LINK затем компонует вместе эти два OBJ-модуля в один выполнимый EXE-модуль.

В PASCAL-программе определены две переменные: temp_row и temp_col, которые содержат введенные с клавиатуры значения строки и колонки соответственно. Программа передает адреса переменных temp_row и temp_col в виде параметров в ассемблерную подпрограмму для установки курсора по этим координатам. PASCAL-программа определяет также имя ассемблерной подпрограммы в операторе procedure как move_cursor и определяет два параметра, как extern (внешние). Оператор в PASCAL-программе, который вызывает ассемблерную программу по имени и передает параметры, имеет следующий вид:

```
move_cursor (temp_row, temp_col);
```

Через стек передаются следующие величины: указатель блока вызывающей программы, указатель на сегмент возврата, смещение возврата и адреса двух передаваемых параметров. Ниже показаны смещения для каждого элемента в стеке:

00 Указатель блока вызывающей программы
02 Указатель сегмента возврата
04 Указатель смещения возврата
06 Адрес второго параметра
08 Адрес первого параметра

Так как ассемблерная подпрограмма будет использовать регистр BP, то его необходимо сохранить в стеке для последующего восстановления при возврате в вызывающую PASCAL-программу. Заметьте, что этот шаг в вызываемой подпрограмме аналогичен предыдущему примеру на рис.21.6.

Регистр SP обычно адресует элементы стека. Но так как этот регистр нельзя использовать в качестве индексного регистра, то после сохранения старого значения регистра BP необходимо переслать адрес из регистра SP в BP. Этот шаг дает возможность использовать регистр BP в качестве индексного регистра для доступа к элементам в стеке.

Следующий шаг - получить доступ к адресам двух параметров в стеке. Первый переданный параметр (адрес строки) находится в стеке по смещению 08, и может быть адресован по BP+08. Второй переданный параметр (адрес столбца) находится в стеке по смещению 06 и может быть адресован по BP+06.

Два адреса из стека должны быть переданы в один из индексных регистров BX, DI или SI. В данном примере адрес строки пересылается из [BP+08] в регистр SI, а затем содержимое из [SI] (значение строки) пересылается в регистр DH.

Значение столбца пересылается аналогичным способом в регистр DL. Затем подпрограмма использует значения строки и столбца в регистре DX при вызове BIOS для установки курсора. При выходе подпрограмма восстанавливает регистр BP. Команда RET имеет операнд, значение которого в два раза больше числа параметров, в данном случае 2x2, или 4. Параметры автоматически выводятся из стека и управление переходит в вызывающую программу.

Если в подпрограмме предстоит изменить сегментный регистр то необходимо сохранить его значение командой PUSH на входе и восстановить командой POP на выходе. Можно также использовать стек для передачи величин

из подпрограммы в вызывающую программу. Хотя рассмотренная подпрограмма не возвращает каких-либо значений, в языке PASCAL предполагается, что подпрограмма возвращает одно слово в регистре AX или двойное слово в регистровой паре DX:AX.

В результате компановки двух программ будет построена карта компановки, в которой первый элемент PASCALL представляет PASCALL-программу, второй элемент CODESEG (имя сегмента кода) представляет ассемблерную подпрограмму. Далее следует несколько подпрограмм для PASCALL-программы. Эта довольно тривиальная программа занимает в результате шест.5720 байт памяти - более 20К. Компилирующие языки обычно генерируют объектные коды значительно превышающие по объему размеры компилируемой программы.

КОМПОНОВКА ПРОГРАММ НА ЯЗЫКЕ СИ И АССЕМБЛЕРЕ

Трудность описания связи программ на языке C и ассемблерных программ состоит в том, что различные версии языка C имеют разные соглашения о связях и для более точной информации следует пользоваться руководством по имеющейся версии языка C. Здесь приведем лишь некоторые соображения, представляющие интерес:

- Большинство версий языка C обеспечивают передачу параметров через стек в обратной (по сравнению с другими языками) последовательности. Обычно доступ, например, к двум параметрам, передаваемым через стек, осуществляется следующим образом:

```
MOV  ES,BP
MOV  BP,SP
MOV  DH,[BP+4]
MOV  DL,[BP+6]
...
POP  BP
RET
```

- Некоторые версии языка C различают прописные и строчные буквы, поэтому имя ассемблерного модуля должно быть представлено в том же символьном регистре, какой используют для ссылки C-программы.

- В некоторых версиях языка C требуется, чтобы ассемблерные программы, изменяющие регистры DI и SI, записывали их содержимое в стек при входе и восстанавливали эти значения из стека при выходе.

- Ассемблерные программы должны возвращать значения, если это необходимо, в регистре AX (одно слово) или в регистровой паре DX:AX (два слова).

- Для некоторых версий языка C, если ассемблерная программа устанавливает флаг DF, то она должна сбросить его командой CLD перед возвратом.

ОСНОВНЫЕ ПОЛОЖЕНИЯ НА ПАМЯТЬ

- В основной программе, вызывающей подпрограмму, необходимо определять точку входа как EXTRN, а в подпрограмме - как PUBLIC.

- Будьте внимательны при использовании рекурсий, когда подпрограмма 1 вызывает подпрограмму 2, которая в свою очередь вызывает подпрограмму 1.

- Если кодовые сегменты необходимо скомпоновать в один сегмент, то необходимо определить их с одинаковыми именами, одинаковыми классами и атрибутом PUBLIC.

- Для простоты программирования начинайте выполнение с основной программы.

- Определение общих данных в основной программе обычно проще (но не обязательно). Основная программа определяет общие данные как PUBLIC, а подпрограмма (или подпрограммы) - как EXTRN.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

21.1. Предположим, что программа MAINPRO должна вызвать под программу SUBPRO. а) Какая директива в программе MAINPRO указывает ассемблеру, что имя SUBPRO определено вне ее собственного кода? б) Какая директива в подпрограмме SUBPRO необходима для того, чтобы имя точки входа было доступно в основной программе MAINPRO?

21.2. Предположим, что в программе MAINPRO определены переменные QTY как DB, VALUE как DW и PRICE как DW. Подпрограмма SUBPRO должна разделить VALUE на QTY и записать частное в PRICE. а) Каким образом программа MAINPRO указывает ассемблеру, что три переменные должны быть доступный извне основной программы? б) Каким образом подпрограмма SUBPRO указывает ассемблеру, что три переменные определены в другом модуле?

21.3. На основании вопросов 21.2 и 21.3 постройте работающую программу и проверьте ее.

21.4. Измените программу из предыдущего вопроса так, чтобы программа MAINPRO передавала все три переменные, как параметры. Подпрограмма SUBPRO должна возвращать результат через параметр.

21.5. Теперь предлагаем упражнение, на которое потребуется больше времени. Требуется расширить программу из вопроса 21.4 так, чтобы программа MAINPRO позволяла вводить количество (QTY) и общую стоимость (VALVE) с клавиатуры, подпрограмма SUBCONV преобразовывала ASCII-величины в двоичное представление; подпрограмма SUBCALC вычисляла цену (PRICE); и подпрограмма SUBDISP преобразовывала двоичную цену в ASCII-представление и выводила результат на экран.

ГЛАВА 22 Программный загрузчик

Ц е л ь: Раскрыть особенности загрузки выполнимых модулей в память для выполнения.

ВВЕДЕНИЕ

В данной главе описана организация базовой версии DOS и операции, которые выполняет DOS для загрузки выполнимых модулей в память для выполнения. DOS состоит из четырех основных программ, которые обеспечивают конкретные функции:

1. Блок начальной загрузки находится на первом секторе нулевой дорожки дискеты DOS, а также на любом диске, форматированном командой FORMAT /S. Когда вы инициируете систему (предполагается, что DOS расположен на дисководе A или C) происходит автоматическая загрузка с диска в память блока начальной загрузки. Этот блок представляет собой программу, которая затем загружает с диска в память три программы, описанные ниже.

2. Программа IBMBIO.COM обеспечивает интерфейс низкого уровня с программами BIOS в ROM; она загружается в память, начиная с адреса шест.00600. При инициализации программа IBMBIO.COM определяет состояние всех устройств и оборудования, а затем загружает программу COMMAND.COM. Программа IBMBIO.COM управляет операциями ввода-вывода между памятью и внешними устройствами, такими как видеомонитор и диск.

3. Программа IBMDOS.COM обеспечивает интерфейс высокого уровня с программами и загружается в память, начиная с адреса шест.00B00. Эта программа управляет оглавлениями и файлами на диске, блокированием и деблокированием дисковых записей, функциями INT 21H, а также содержит ряд других сервисных функций.

4. Программа COMMAND.COM выполняет различные команды DOS, такие как DIR или CHKDSK, а также выполняет COM, EXE и BAT-программы. Она состоит из трех частей: небольшая резидентная часть, часть инициализации и транзитная часть. Программа COMMAND.COM, подробно рассмотренная в следующем разделе, отвечает за загрузку выполняемых программ с диска в память.

На рис.22.1 показана карта распределения памяти. Некоторые элементы могут отличаться в зависимости от модели компьютера.

```
Начальный            Программа
адрес

00000  Векторная таблица прерываний (см.гл.23)
00400  Область связи с ROM (ПЗУ)
00500  Область связи с DOS
00600  IBMBIO.COM
XXXX0  IBMDOS.COM
       Буфер каталога
       Дисковый буфер
       Таблица параметров дисковода или таблица распределения файлов
       (FAT, по одной для каждого дисковода)
XXXX0  Резидентная часть COMMAND.COM
XXXX0  Внешние команды или утилиты (COM или EXE-файлы)
XXXX0  Пользовательский стек для COM-файлов (256 байтов)
XXXX0  Транзитная часть COMMAND.COM, записывается в самые старшие
       адреса памяти.
```

Рис.22.1. Карта распределения DOS в памяти.

КОМАНДНЫЙ ПРОЦЕССОР COMMAND.COM

Система загружает три части программы COMMAND.COM в память во время сеанса работы постоянно или временно. Ниже описано назначение каждой из трех частей COMMAND.COM:

1. Резидентная часть непосредственно следует за программой

IBMDOS.COM (и ее области данных), где она находится на протяжении всего сеанса работы. Резидентная часть обрабатывает все ошибки дисковых операций ввода-вывода и управляет следующими прерываниями:

INT 22H Адрес программы обработки завершения задачи.
INT 23H Адрес программы реакции на Ctrl/Break.
INT 24H Адрес программы реакции на ошибки дисковых операций чтения/записи или сбойный участок памяти в таблице распределения файлов (FAT).
INT 27H Завершение работы, после которого программа остается резидентной.

2. Часть инициализации непосредственно следует за резидентной частью и содержит средства поддержки AUTOEXEC-файлов. В начале работы системы данная часть первой получает управление. Она выдает запрос на ввод даты и определяет сегментный адрес, куда система должна загружать программы для выполнения. Ни одна из этих программ инициализации не потребуются больше во время сеанса работы. Поэтому первая же команда вводимая с клавиатуры и вызывающая загрузку некоторой программы с диска перекрывают часть инициализации в памяти.

3. Транзитная часть загружается в самые старшие адреса памяти. "Транзит" обозначает, что DOS может перекрыть данную область другими программами, если потребуется. Транзитная часть программы COMMAND.COM выводит на экран приглашение DOS A> или C>, вводит и выполняет запросы. Она содержит настраивающий загрузчик и предназначена для загрузки COM- или EXE-файлов с диска в память для выполнения. Если поступил запрос на выполнение какой-либо программы, то транзитная часть строит префикс программного сегмента (PSP) непосредственно вслед за резидентной частью COMMAND.COM. Затем она загружает запрошенную программу с диска в память по смещению шест.100 от начала программного сегмента, устанавливает адреса выхода и передает управление в загруженную программу. Ниже приведена данная последовательность:

IBMBIO.COM
IBMDOS.COM
COMMAND.COM (резидент)
Префикс программного сегмента
Выполняемая программа
...
COMMAND.COM (транзитная часть, может быть перекрыта).

Выполнение команды RET или INT 20H в конце программы приводит к возврату в резидентную часть COMMAND.COM. Если транзитная часть была перекрыта, то резидентная часть перезагружает транзитную часть с диска в память.

ПРЕФИКС ПРОГРАММНОГО СЕГМЕНТА

Префикс программного сегмента (PSP) занимает 256 (100H) байт и всегда предшествует в памяти каждой COM- или EXE-программе, которая должна быть выполнена. PSP содержит следующие поля:

00 Команда INT 20H (CD20).
02 Общий размер доступной памяти в формате xxxx0. Например, 512K указывается как 8000H вместо 80000H.
04 Зарезервировано.
05 Длинный вызов диспетчера функций DOS.

0A Адрес подпрограммы завершения.
0E Адрес подпрограммы реакции на Ctrl/Break.
12 Адрес подпрограммы реакции на фатальную ошибку.
16 Зарезервировано.
2C Сегментный адрес среды для хранения ASCIIZ строк.
50 Вызов функций DOS (INT 21H и RETF).
5C Параметрическая область 1, форматированная как стандартный неоткрытый блок управления файлов (FCB#1).
6C Параметрическая область 2, форматированная как стандартный неоткрытый блок управления файлом (FCB#2); перекрывается, если блок FCB#1 открыт.
80-FF Буфер передачи данных (DTA).

Буфер передачи данных DTA

Данная часть PSP начинается по адресу 80H и представляет собой буферную область ввода-вывода для текущего дисковода. Она содержит в первом байте число, указывающее сколько раз были нажаты клавиши на клавиатуре непосредственно после ввода имени программы. Начиная со второго байта, находятся введенные символы (если таковые имеются). Далее следует всевозможный "мусор", оставшийся в памяти после работы предыдущей программы. Следующие примеры демонстрируют назначение буфера DTA:

П р и м е р 1. Команда без операндов. Предположим, что вы вызвали программу CALCIT.EXE для выполнения с помощью команды CALCIT [return]. После того, как DOS построит PSP для этой программы, он установит в буфере по адресу шест.80 значение шест.000D. Первый байт содержит число символов, введенных с клавиатуры после имени CALCIT, исключая символ "возврат каретки". Так как кроме клавиши Return не было нажато ни одной, то число символов равно нулю. Второй байт содержит символ возврата каретки, шест.0D. Таким образом, по адресам шест.80 и 81 находятся 000D.

П р и м е р 2. Команда с текстовым операндом. Предположим, что после команды был указан текст (но не имя файла), например, COLOR BY, обозначающий вызов программы COLOR и передачу этой программе параметра "BY" для установки голубого цвета на желтом фоне. В этом случае, начиная с адреса шест.80, DOS установит следующие значения байт:

```
80:  03 20 42 59 0D
```

Эти байты обозначают длину 3, пробел, "BY" и возврат каретки.

П р и м е р 3. Команда с именем файла в операнде. Программы типа DEL (удаление файла) предполагают после имени программы ввод имени файла в качестве параметра. Если будет введено, например, DEL B:CALCIT.OBJ [return], то PSP, начиная с адресов шест.5C и шест.80, будет содержать:

```
5C:  02 43 41 4C 43 49 54 20 20 4F 42 4A
        C  A  L  C  I  T        O  B  J
80:  0D 20 42 3A 43 41 4C 43 49 54 2E 4F 42 4A 0D
           B  :  C  A  L  C  I  T  .  0  B  J
```

Начиная с адреса шест.5C, находится неоткрытый блок FCB, содержащий имя файла, который был указан в параметре, CALCIT.OBJ, но не имя выполняемой программы. Первый символ указывает номер дисковода (02=B в данном случае). Следом за CALCIT находятся два пробела, которые дополняют имя файла до восьми символов, и тип файла, OBJ. Если ввести два параметра, например:

```
progname A:FILEA,B:FILEB
```

тогда DOS построит FCB для FILEA по смещению шест.5C и FCB для FILEB по смещению шест.6C.

Начиная с адреса шест.80 в этом случае содержится число введенных символов (длина параметров) - 16, пробел (шест.20) A:FILEA,B:FILEB и символ возврат каретки (OD).

Так как PSP непосредственно предшествует вашей программе, то возможен доступ к области PSP для обработки указанных файлов или для предпринятия определенных действий. Для локализации буфера DTA COM-программа может просто поместить шест.80 в регистр SI и получить доступ следующим образом:

```
MOV  SI,80H             ;Адрес DTA
CMP  BYTE PTR [SI],0    ;В буфере нуль?
JE   EXIT
```

Для EXE-программы нельзя с уверенностью утверждать, что кодовый сегмент непосредственно располагается после PSP. Однако, здесь при инициализации регистры DS и ES содержат адрес PSP, так что можно сохранить содержимое регистра ES после загрузки регистра DS:

```
MOV  AX,DSEG
MOV  DS,AX
MOV  SAVEPSP,ES
```

Позже можно использовать сохраненный адрес для доступа к буферу PSP:

```
MOV  SI,SAVEPSP
CMP  BYTE PTR [SI+ 80H],0     ;В буфере нуль?
JE   EXIT
```

DOS версии 3.0 и старше содержит команду INT 62H, загружающую в регистр BX адрес текущего PSP, который можно использовать для доступа к данным в PSP.

ВЫПОЛНЕНИЕ COM-ПРОГРАММЫ

В отличие от EXE-файла, COM-файл не содержит заголовок на диске. Так как организация COM-файла намного проще, то для DOS необходимо "знать" только то, что тип файла - COM.

Как описано выше, загруженным в память COM- и EXE-файлам предшествует префикс программного сегмента. Первые два байта этого префикса содержат команду INT 20H (возврат в DOS). При загрузке COM-программы DOS устанавливает в четырех сегментных регистрах адрес первого байта PSP. Затем устанавливается указатель стека на конец 64 Кбайтового сегмента (шест.FFFE) или на конец памяти, если сегмент не достаточно большой. В вершину стека заносится нулевое слово. В командный указатель помещается шест.100 (размер PSP). После этого управление передается по адресу регистровой пары CS:IP, т.е. на адрес непосредственно после PSP. Этот адрес является началом выполняемой COM-программы и должен содержать выполнимую команду.

При выходе из программы команда RET заносит в регистр IP нулевое слово, которое было записано в вершину стека при инициализации. В этом случае в регистровой паре CS:IP получается адрес первого байта PSP, где находится команда INT 20H. При выполнении этой команды управление передается в резидентную часть COMMAND.COM. (Если программа завершается по команде INT 20H вместо RET, то управление непосредственно передается в COMMAND.COM).

ВЫПОЛНЕНИЕ EXE-ПРОГРАММЫ

EXE-модуль, созданный компановщиком, состоит из следующих двух частей: 1) заголовок - запись, содержащая информацию по управлению и настройке программы и 2) собственно загрузочный модуль.

В заголовке находится информация о размере выполняемого модуля, области загрузки в памяти, адресе стека и относительных смещениях, которые должны заполнить машинные адреса в соответствии с относительными шест. позициями:

00 Шест.4D5A. Компоновщик устанавливает этот код для идентификации правильного EXE-файла.
02 Число байтов в последнем блоке EXE-файла.
04 Число 512 байтовых блоков EXE-файла, включая заголовок.
06 Число настраиваемых элементов.
08 Число 16-тибайтовых блоков (параграфов) в заголовке, (необходимо для локализации начала выполняемого модуля, следующего после заголовка).
0A Минимальное число параграфов, которые должны находится после загруженной программы.
0C Переключатель загрузки в младшие или старшие адреса. При компановке программист должен решить, должна ли его программа загружаться для выполнения в младшие адреса памяти или в старшие Обычным является загрузка в младшие адреса. Значение шест.0000 указывает на загрузку в старшие адреса, а шест.FFFF - в младшие. Иные значения определяют максимальное число параграфов, которые должны находиться после загруженной программы.
0E Относительный адрес сегмента стека в выполняемом модуле.
10 Адрес,который загрузчик должен поместить в регистр SP перед передачей управления в выполнимый модуль.
12 Контрольная сумма - сумма всех слов в файле (без учета переполнений) используется для проверки потери данных.
14 Относительный адрес, который загрузчик должен поместить в регистр IP до передачи управления в выполняемый модуль.
16 Относительный адрес кодового сегмента в выполняемом модуле. Этот адрес загрузчик заносит в регистр CS.
18 Смещение первого настраиваемого элемента в файле.
1A Номер оверлейного фрагмента: нуль обозначает, что заголовок относится к резидентной части EXE-файла.
1C Таблица настройки, содержащая переменное число настраиваемых элементов, соответствующее значению по смещению 06.

Заголовок имеет минимальный размер 512 байтов и может быть больше, если программа содержит большое число настраиваемых элементов. Позиция 06 в заголовке указывает число элементов в выполняемом модуле, нуждающихся в настройке. Каждый элемент настройки в таблице, начинающейся в позиции 1C заголовка, состоит из двухбайтовых величин смещений и двухбайтовых сегментных значений.

Система строит префикс программного сегмента следом за резидентной частью COMMAND.COM, которая выполняет операцию загрузки. Затем COMMAND.COM выполняет следующие действия:

- Считывает форматированную часть заголовка в память.
- Вычисляет размер выполнимого модуля (общий размер файла в позиции 04 минус размер заголовка в позиции 08) и загружает модуль в память с начала сегмента.
- Считывает элементы таблицы настройки в рабочую область и прибавляет значения каждого элемента таблицы к началу сегмента (позиция OE).
- Устанавливает в регистрах SS и SP значения из заголовка и прибавляет адрес начала сегмента.
- Устанавливает в регистрах DS и ES сегментный адрес префикса

программного сегмента.

- Устанавливает в регистре CS адрес PSP и прибавляет величину смещения в заголовке (позиция 16) к регистру CS. Если сегмент кода непосредственно следует за PSP, то смещение в заголовке равно 256 (шест.100). Регистровая пара CS:IP содержит стартовый адрес в кодовом сегменте, т.е. начальный адрес программы.

После инициализации регистры CS и SS содержат правильные адреса, а регистр DS (и ES) должны быть установлены в программе для их собственных сегментов данных:

```
1.  PUSH  DS              ;Занести адрес PSP в стек
2.  SUB   AX,AX           ;Занести нулевое значение в стек
3.  PUSH  AX              ; для обеспечения выхода из программы
4.  MOV   AX,datasegname  ;Установка в регистре DX
5.  MOV   DS,AX           ; адреса сегмента данных
```

При завершении программы команда RET заносит в регистр IP нулевое значение, которое было помещено в стек в начале выполнения программы. В регистровой паре CS:IP в этом случае получается адрес, который является адресом первого байта PSP, где расположена команда INT 20H. Когда эта команда будет выполнена, управление перейдет в DOS.

ПРИМЕР EXE-ПРОГРАММЫ

Рассмотрим следующую таблицу компановки (MAP) программы:

```
Start     Stop      Length    Name      Class
00000H    0003AH    003BH     CSEG      CODE
00040H    0005AH    001BH     DSEG      DATA
00060H    0007FH    0020H     STACK     STACK
Program  entry point at 0000:0000
```

Таблица MAP содержит относительные (не действительные) адреса каждого из трех сегментов. Символ H после каждого значения указывает на шестнадцатиричный формат. Заметим, что компоновщик может организовать эти сегменты в последовательности отличного от того, как они были закодированы в программе.

В соответствии с таблицей MAP кодовый сегмент CSEG находится по адресу 00000 - этот относительный адрес является началом выполняемого модуля. Длина кодового сегмента составляет шест.003B байтов. Следующий сегмент по имени DSEG начинается по адресу шест.00040 и имеет длину шест.001B. Адрес шест.00040 является первым после CSEG адресом, выровненным на границу параграфа (т.е. это значение кратно шест.10). Последний сегмент, STACK, начинается по адресу шест.00060 - первому после DSEG, адресу выровненному на границу параграфа.

С помощью отладчика DEBUG нельзя проверить содержимое заголовка, так как при загрузке программы для выполнения DOS замещает заголовок префиксом программного сегмента. Однако, на рынке программного обеспечения имеются различные сервисные утилиты (или можно написать собственную), которые позволяют просматривать содержимое любого дискового сектора в шестнадцатиричном формате. Заголовок для рассматриваемого примера программы содержит следующую информацию (содержимое слов представлено в обратной последовательности байтов):

00 Шест.4D5A.
02 Число байтов в последнем блоке: 5B00.
04 Число 512 байтовых блоков в файле, включая заголовок: 0200 (шест.0002 x 512 = 1024).

06 Число элементов в таблице настройки, находящейся после форматированной части заголовка: 0100, т.е. 0001.
08 Число 16 байтовых элементов в заголовке: 2000 (шест.0020=32 и 32x16=512).
0C Загрузка в младшие адреса: шест.FFFF.
0E Относительный адрес стекового сегмента: 6000 или шест.60.
10 Адрес для загрузки в SP: 2000 или шест.20.
14 Смещение для IP: 0000.
16 Смещение для CS: 0000.
18 Смещение для первого настраиваемого элемента: 1E00 или шест.1E.

После загрузки программы под управлением отладчика DEBUG регистры получают следующие значения:

```
SP = 0020     DS = 138F     ES = 138F
SS = 13A5     CS = 139F     IP = 0000
```

Для EXE-модулей загрузчик устанавливает в регистрах DS и ES адрес префикса программного сегмента, помещенного в доступной области памяти, а в регистрах IP, SS и SP - значения из заголовка программы.

Регистр SP

Загрузчик использует шест.20 из заголовка для инициализации указателя стека значением длины стека. В данном примере стек был определен, как 16 DUP (?), т.е. 16 двухбайтовых полей общей длиной 32 (шест.20) байта. Регистр SP указывает на текущую вершину стека.

Регистр CS

В соответствии со значением в регистре DS после загрузки программы, адрес PSP равен шест.138F(0). Так как PSP имеет длину шест.100 байтов, то выполняемый модуль, следующий непосредственно после PSP, находится по адресу шест.138F0+100=139F0. Это значение устанавливается загрузчиком в регистре CS. Таким образом, регистр CS определяет начальный адрес кодовой части программы (CSEG). С помощью команды D CS:0000 в отладчике DEBUG можно просмотреть в режиме дампа машинный код в памяти. Обратите внимание на идентичность дампа и шестнадцатиричной части ассемблерного LST файла кроме операндов, отмеченных символом R.

Регистр SS

Для установки значения в регистре SS загрузчик также использует информацию из заголовка:

```
Начальный адрес PSP (см.DS)   138F0
Длина PSP                       100
Относительный адрес стека        60
                              -----
Адрес стека                   13A50
```

Регистр DS

Загрузчик использует регистр DS для установки начального адреса PSP. Так как заголовок не содержит стартового адреса, то регистр DS необходимо инициализировать в программе следующим образом:

```
0004 B8 ---- R      MOV  AX,DSEG
0007 8E D8          MOV  DS,AX
```

Ассемблер оставляет незаполненным машинный адрес сегмента DSEG, который становится элементом таблицы настройки в заголовке. С помощью отладчика DEBUG можно просмотреть завершенную команду в следующем виде:

```
B8 A313
```

Значение A313 загружается в регистр DS в виде 13A3. В результате имеем

Регистр	Адрес	Смещение
CS	139F0	00
DS	13A30	40
SS	13A50	60

В качестве упражнения выполните трассировку любой вашей скомпонованной программы под управлением отладчика DEBUG и обратите внимание на изменяющиеся значения в регистрах:

Команда	Изменяющиеся регистры
PUSH DS	IP и SP
SUB AX,AX	IP и AX (если был не нуль)
PUSH AX	IP и SP
MOV AX,DSEG	IP и AX
MOV DS,AX	IP и DS

Регистр DS содержит теперь правильный адрес сегмента данных. Можно использовать теперь команду D DS:00 для просмотра содержимого сегмента данных DSEG и команду D SS:00 для просмотра содержимого стека.

ФУНКЦИИ ЗАГРУЗКИ И ВЫПОЛНЕНИЯ ПРОГРАММЫ

Рассмотрим теперь, как можно загрузить и выполнить программу из другой программы. Функция шест.4B дает возможность одной программе загрузить другую программу в память и при необходимости выполнить. Для этой функции необходимо загрузить адрес ASCIIZ-строки в регистр DX, а адрес блока параметров в регистр BX (в действительности в регистровую пару ES:BX). В регистре AL устанавливается номер функции 0 или 3:

AL=0. З а г р у з к а и в ы п о л н е н и е. Данная операция устанавливает префикс программного сегмента для новой программы, а также адрес подпрограммы реакции на Cntrl/Break и адрес передачи управления на следующую команду после завершения новой программы. Так как все регистры, включая SP, изменяют свои значения, то данная операция не для новичков. Блок параметров, адресуемый по ES:BX, имеет следующий формат:

Смещение	Назначение
0	Двухбайтовый сегментный адрес строки параметров для передачи.
2	Четырехбайтовый указатель на командную строку в PSP+80H.
6	Четырехбайтовый указатель на блок FCB в PSP+5CH.
10	Четырехбайтовый указатель на блок FCB в PSP+6CH.

AL=3. О в е р л е й н а я з а г р у з к а. Данная операция загружает программу или блок кодов, но не создает PSP и не начинает выполнение.

Таким образом можно создавать оверлейные программы. Блок параметров адресуется по регистровой паре ES:BX и имеет следующий формат:

Смещение	Назначение
0	Двухбайтовый адрес сегмента для загрузки файла.
2	Двухбайтовый фактор настройки загрузочного модуля.

Возможные коды ошибок, возвращаемые в регистре AX: 01, 02, 05, 08, 10 и 11. Программа на рис.22.2 запрашивает DOS выполнить команду DIR для дисковода D. Выполните эту программу, как EXE-модуль. (Автор благодарен журналу PC Magazine за эту идею).

```
TITLE   EXDOS   (EXE) Функция DOS 4BH для выполнения DIR
CSEG    GMENT   PARA 'Code'
        ASSUME  CS:CSEG,DS:CSEG,ES:CSEG
BEGIN:  JMP     SHORT MAIN
;--------------------------------------------------------
PARAREA DW      ?               ;Адрес строки вызова
        DW      OFFSET DIRCOM   ;Указатель
                                ; на командную строку
        DW      CSEG
        DW      OFFSET FCB1     ;Указатель на FCB2
        DW      CSEG
DIRCOM  DB      17,'/C DIR D:',13,0
FCB1    DB      16 DUP(0)
FCB2    DB      16 DUP(0)
PROGNAM DB      'D:COMMAND.COM',0
; --------------------------------------------------------
MAIN    PROC    FAR
        MOV     AH,4AH          ;Получить 64K памяти
        MOV     BH,100H         ; в параграфах
        INT     21H
        JC      E10ERR          ;Нет памяти?
        MOV     DI,2CH          ;Получить сегментный адрес
        MOV     AX,[DI]         ; строки вызова
        LEA     SI,PARAREA      ; и записать его в
        MOV     [SI],AX         ; 1 слово блока параметров
        MOV     AX,CS           ;Загрузить в DS и ES
        MOV     DS,AX           ; адрес CSEG
        MOV     ES,AX
        MOV     AH,4BH          ;Функция загрузки
        MOV     AL,00           ; и выполнения
        LEA     BX,PARAREA      ; COMMAND.COM
        LEA     DX,PROGNAM
        INT     21H             ;Вызвать DOS
        JC      E20ERR          ;Ошибка выполнения?
        MOV     AL,00           ;Нет кода ошибки
        JMP     X10XIT
E10ERR:
        MOV     AL,01           ;Код ошибки 1
        JMP     X10XIT
E20ERR:
        MOV     AL,02           ;Код ошибки 2
        JMP     X10XIT
E10XIT:
        MOV     AH,4CH          ;Функция завершения
        INT     21H             ;Вызвать DOS
MAIN    ENDP
```

```
CSEG    ENDS
        END
```

Рис.22.2. Выполнение команды DIR из программы.

ГЛАВА 23 Прерывания BIOS и DOS

Ц е л ь: Описать функции, доступные через прерывания BIOS и DOS.

ВВЕДЕНИЕ

Прерывание представляет собой операцию, которая приостанавливает выполнение программ для специальных системных действий. Необходимость прерываний обусловлено двумя основными причинами: преднамеренный запрос таких действий, как операции ввода-вывода на различные устройства и непредвиденные программные ошибки (например, переполнение при делении).

Система BIOS (Basic Input/Output System) находится в ROM и управляет всеми прерываниями в системе. В предыдущих главах уже использовались некоторые прерывания для вывода на экран дисковых операций ввода-вывода и печати. В этой главе описаны различные BIOS- и DOS-прерывания, резидентные программы и команды IN и OUT.

ОБСЛУЖИВАНИЕ ПРЕРЫВАНИЙ

В компьютерах IBM PC ROM находится по адресу FFFF0H. При включении компьютера процессор устанавливает состояние сброса, выполняет контроль четности, устанавливает в регистре CS значение FFFFH, а в регистре IP - нуль. Первая выполняемая команда поэтому находится по адресу FFFF:0 или FFFF0, что является точкой входа в BIOS. BIOS проверяет различные порты компьютера для определения и инициализации подключенных устрой ств. Затем BIOS создает в начале памяти (по адресу 0) таблицу прерываний, которая содержит адреса обработчиков прерываний, и выполняет две операции INT 11H (запрос списка присоединенного оборудования) и INT 12H (запрос размера физической памяти).

Следующим шагом BIOS определяет имеется ли на диске или дискете операционная система DOS. Если обнаружена системная дискета, то BIOS выполняет прерывание INT 19H для доступа к первому сектору диска, содержащему блок начальной загрузки. Этот блок представляет собой программу, которая считывает системные файлы IBMBIO.COM, IBMDOS.COM и COMMAND.COM с диска в память. После этого память имеет следующее распределение:

Таблица векторов прерываний
Данные BIOS
IBMBIO.COM и IBMDOS.COM
Резидентная часть COMMAND.COM
Доступная память для прикладных программ
Транзитная часть COMMAND.COM
Конец RAM (ОЗУ)
ROM BASIC
ROM BIOS

Внешние устройства передают сигнал внимания через контакт INTR в

процессор. Процессор реагирует на этот запрос, если флаг прерывания IF установлен в 1 (прерывание разрешено), и (в большинстве случаев) игнорирует запрос, если флаг IF установлен в 0 (прерывание запрещено).

Операнд в команде прерывания, например, INT 12H, содержит тип прерывания, который идентифицирует запрос. Для каждого типа система содержит адрес в таблице векторов прерываний, начинающейся по адресу 0000. Так как в таблице имеется 256 четырехбайтовых элементов, то она занимает первые 1024 байта памяти от шест.0 до шест.3FF. Каждый элемент таблицы указывает на подпрограмму обработки указанного типа прерывания и содержит адрес кодового сегмента и смещение, которые при прерывании устанавливаются в регистры CS и IP соответственно. Список элементов таблицы векторов прерываний приведен на рис. 23.1.

Прерывание заносит в стек содержимое флагового регистра, регистра CS и регистра IP. Например, для прерывания 12H (которое возвращает в регистре AX размер памяти) адрес элемента таблицы равен шест.0048 (шест.12 x 4 = шест.48). Операция выделяет четырехбайтовый элемент по адресу шест. 0048 и заносит два байта в регистр IP и два байта в регистр SS. Адрес, который получается в регистровой паре CS:IP, представляет собой адрес начала

подпрограммы в области BIOS, которая получает управление. Возврат из этой подпрограммы осуществляется командой IRET (Interrupt Return), которая восстанавливает флаги и регистры CS и IP из стека и передает управление на команду, следующую за выполненной командой прерывания.

Адрес (шест.)		Функция прерываний (шест.)
0-3	0	Деление на нуль
4-7	1	Пошаговый режим (трассировка DEBUG)
8-B	2	Немаскированное прерывание (NMI)
C-F	3	Точка останова (используется в DEBUG)
10-13	4	Переполнение регистра
14-17	5	Печать экрана
18-1F	6,7	Зарезервировано
20-23	8	Сигнал от таймера
24-27	9	Сигнал от клавиатуры
28-37	A,B,C,D	Используются в компьютерах AT
38-3B	E	Сигнал от дискетного дисковода
3C-3F	F	Используется для принтера
40-43	10	Управление дисплеем (см.гл. 8, 9, 10)
44-47	11	Запрос оборудования (см.гл.9)
48-4B	12	Запрос размера памяти (см.гл.2)
4C-4F	13	Дисковые операции ввода-вывода (см.гл.18)
50-53	14	Управление коммуникационным адаптером
54-57	15	Кассетные операции и спец. функции AT
58-5B	16	Ввод с клавиатуры (см.гл.9)
5C-5F	17	Вывод на принтер (см.гл.19)
60-63	18	Обращение к BASIC, встроенному в ROM
64-67	19	Перезапуск системы
68-6B	1A	Запрос и установка времени и даты
6C-6F	1B	Прерывание от клавиатуры
70-73	1C	Прерывание от таймера
74-77	1D	Адрес таблицы параметров дисплея
78-7B	1E	Адрес таблицы параметров дисковода
7C-7F	1F	Адрес таблицы графических символов
80-83	20	Нормальное завершение программы (DOS)
84-87	21	Обращение к функциям DOS
88-8B	22	Адрес обработки завершения задачи (DOS)
8C-8F	23	Адрес реакции по Ctrl/Break (DOS)

90-93	24	Адрес реакции на фатальную ошибку (DOS)
94-97	25	Абсолютное чтение с диска (DOS)
98-9B	26	Абсолютная запись на диск (DOS)
97-9F	27	Создание резидентной программы (DOS)
AO-FF	28-3F	Другие функции DOS
100-1FF	40-7F	Зарезервировано
200-217	80-85	Зарезервировано для BASIC
218-3C3	86-F0	Используются BASIC-интерпретатором
3C4-3FF	F1-FF	Зарезервировано

Примечание: Прерывания 00-1F относятся к BIOS,
прерывания 20-FF относятся к DOS и BASIC.

Рис.23.1. Таблица адресов прерываний.

ПРЕРЫВАНИЯ BIOS

В данном разделе приведены основные прерывания BIOS.

INT 05H. П е ч а т ь э к р а н а. Выполняет вывод содержимого экрана на печатающее устройство. Команда INT 05H выполняет данную операцию из программы, а нажатие клавишей Ctrl/PrtSc - с клавиатуры. Операция запрещает прерывания и сохраняет позицию курсора.

INT 10H. У п р а в л е н и е д и с п л е е м. Обеспечивает экранные и клавиатурные операции, детально описанные в гл.9.

INT 11H. З а п р о с с п и с к а п р и с о е д и н е н н о г о о б о р у д о в а н и я. Определяет наличие различных устройств в системе, результирующее значение возвращает в регистре AX. При включении компьютера система выполняет эту операцию и сохраняет содержимое AX в памяти по адресу шест.410. Значения битов в регистре AX:

Бит	Устройство
15,14	Число подключенных принтеров.
13	Последовательный принтер.
12	Игровой адаптер.
11-9	Число последовательных адаптеров стыка RS232.
7,6	Число дискетных дисководов, при бите 0=1: 00=1, 01=2, 10=3 и 11=4.
5,4	Начальный видео режим: 00 = не используется, 01 = 40x25 плюс цвет, 10 = 80x25 плюс цвет, 11 = 80x25 черно-белый режим.
1	Значение 1 говорит о наличии сопроцессора.
0	Значение 1 говорит о наличии одного или более дисковых устройств и загрузка операционной системы должна осуществляться с диска.

INT 12H З а п р о с р а з м е р а ф и з и ч е с к о й п а м я т и. Возвращает в регистре AX размер памяти в килобайтах, например, шест.200 соответствует памяти в 512 К. Данная операция полезна для выравнивания размера программы в соответствии с доступной памятью.

INT 13H. Д и с к о в ы е о п е р а ц и и в в о д а - в ы в о д а. Обеспечивает операции ввода-вывода для дискет и винчестера, рассмотренные

в главе 16.

INT 14H. У п р а в л е н и е к о м м у н и к а ц и о н н ы м а д а п т е р о м. Обеспечивает последовательный ввод-вывод через коммуникационный порт RS232. Регистр DX должен содержать номер (0 или 1) адаптера стыка RS232. Четыре типа операции, определяемые регистром AH, выполняют прием и передачу символов и возвращают в регистре AX байт состояния коммуникационного порта.

INT 15H. К а с с е т н ы е о п е р а ц и и в в о д а - в ы в о д а и с п е ц и а л ь н ы е ф у н к ц и и д л я к о м п ь ю т е р о в AT. Обеспечивает операции ввода-вывода для кассетного магнитофона, а также расширенные операции для компьютеров AT.

INT 16H. В в о д с к л а в и а т у р ы. Обеспечивает три типа команд ввода с клавиатуры, подробно описанные в гл.9.

INT 17H. В ы в о д н а п р и н т е р. Обеспечивает вывод данных на печатающее устройство. Подробно рассмотрено в гл.19.

INT 18H. О б р а щ е н и е к BASIC, в с т р о е н н о м у в ROM. Вызывает BASIC-интерпретатор, находящийся в постоянной памяти ROM.

INT 19H. П е р е з а п у с к с и с т е м ы. Данная операция при доступном диске считывает сектор 1 с дорожки 0 в область начальной загрузки в памяти (сегмент 0, смещение 7C00) и передает управление по этому адресу. Если дисковод не доступен, то операция передает управление через INT 18H в ROM BASIC. Данная операция не очищает экран и не инициализирует данные в ROM BASIC, поэтому ее можно использовать из программы.

INT 1AH. З а п р о с и у с т а н о в к а т е к у щ е г о в р е м е н и и д а т ы. Считывает и записывает показание часов в соответствии со значением в регистре AH. Для определения продолжительности выполнения программы можно перед началом выполнения установить часы в 0, а после считать текущее время. Отсчет времени идет примерно 18,2 раза в секунду. Значение в регистре AH соответствует следующим операциям:

AH=00 Запрос времени. В регистре CX устанавливается старшая часть значения, а в регистре DX - младшая. Если после последнего запроса прошло 24 часа, то в регистре AL будет не нулевое значение.
AH=01 Установка времени. Время устанавливается по регистрам CX (старшая часть значения) и DX (младшая часть значения).

Коды 02 и 06 управляют временем и датой для AT.

INT 1FH. А д р е с т а б л и ц ы г р а ф и ч е с к и х с и м в о л о в. В графическом режиме имеется доступ к символам с кодами 128-255 в 1К таблице, содержащей по восемь байт на каждый символ. Прямой доступ в графическом режиме обеспечивается только к первым 128 ASCII-символам (от 0 до 127).

ПРЕРЫВАНИЯ DOS

Во время своей работы BIOS использует два модуля DOS: IBMBIO.COM и IBMDOS.COM. Так как модули DOS обеспечивают большое количество разных дополнительных проверок, то операция DOS обычно проще в использовании и менее машиннозависимы, чем их BIOS аналоги.
Модуль IBMBIO.COM обеспечивает интерфейс с BIOS низкого уровня. Эта

программа выполняет управление вводом-выводом при чтении данных из внешних устройств в память и записи из памяти на внешние устройства.

Модуль IBMDOS.COM содержит средства управления файлами и ряд сервисных функций, таких как блокирование и деблокирование записей. Когда пользовательская программа выдает запрос INT 21H, то в программу IBMDOS через регистры передается определенная информация. Затем программа IBMDOS транслирует эту информацию в один или несколько вызовов IBMBIO, которая в свою очередь вызывает BIOS. Указанные связи приведены на следующей схеме:

```
Пользовательский Высший      Низший     ROM     Внешний
уровень          уровень     уровень            уровень
------------¬ -----------¬ -----------¬
¦Программный¦ ¦   DOS    ¦ ¦   DOS    ¦ -----¬ -----------¬
¦запрос в/в ¦¦IBMDOS.COM¦¦IBMBIO.COM¦¦BIOS¦¦Устройство¦
L------------ L----------- L----------- L----- L-----------
```

Как показано выше, прерывания от шест.20 до шест.62 зарезервированы для операций DOS. Ниже приведены наиболее основные из них:

INT 20H. З а в е р ш е н и е п р о г р а м м ы. Запрос завершает выполнение программы и передает управление в DOS. Данный запрос обычно находится в основной процедуре.

INT 21H. З а п р о с ф у н к ц и й DOS. Основная операция DOS, вызывающая определенную функцию в соответствии с кодом в регистре AH. Назначение функций DOS описано в следующем разделе.

INT 22H. А д р е с п о д п р о г р а м м ы о б р а б о т к и з а в е р ш е н и я з а д а ч и. (см.INT 24H).

INT 23H. А д р е с п о д п р о г р а м м ы р е а к ц и и н а Ctrl/Break. (см.INT 24H).

INT 24H. А д р е с п о д п р о г р а м м ы р е а к ц и и н а ф а т а л ь н у ю о ш и б к у. В этом элементе и в двух предыдущих содержатся адреса, которые инициализируются системой в префиксе программного сегмента и, которые можно изменить для своих целей. Подробности приведены в техническом описании DOS.

INT 25H. А б с о л ю т н о е ч т е н и е с д и с к а. См.гл.17.

INT 26H. А б с о л ю т н а я з а п и с ь н а д и с к. См.гл.17.

INT 27H.З а в е р ш е н и е п р о г р а м м ы, о с т а в л я ю щ е е е е р е з е д е н т н о й. Позволяет сохранить COM-программу в памяти. Подробно данная операция рассмотрена в последующем разделе "Резидентные программы".

ФУНКЦИИ ПРЕРЫВАНИЯ DOS INT 21H

Ниже приведены базовые функции для прерывания DOS INT 21H. Код функции устанавливается в регистре AH:

00 Завершение программы (аналогично INT 20H).
01 Ввод символа с клавиатуры с эхом на экран.
02 Вывод символа на экран.
03 Ввод символа из асинхронного коммуникационного канала.
04 Вывод символа на асинхронный коммуникационный канал.
05 Вывод символа на печать (гл.19).

06 Прямой ввод с клавиатуры и вывод на экран.
07 Ввод с клавиатуры без эха и без проверки Ctrl/Break.
08 Ввод с клавиатуры без эха с проверкой Ctrl/Break.
09 Вывод строки символов на экран (гл.8).
0A Ввод с клавиатуры с буферизацией (гл.8).
0B Проверка наличия ввода с клавиатуры.
0C Очистка буфера ввода с клавиатуры и запрос на ввод.
0D Сброс диска (гл.16).
0E Установка текущего дисковода (гл.16).
0F Открытие файла через FCB (гл.16).
10 Закрытие файла через FCB (гл.16).
11 Начальный поиск файла по шаблону (гл.16).
12 Поиск следующего файла по шаблону (гл.16).
13 Удаление файла с диска (гл.16).
14 Последовательное чтение файла (гл.16).
15 Последовательная запись файла (гл.16).
16 Создание файла (гл.16).
17 Переименование файла (гл.16).
18 Внутренняя операция DOS.
19 Определение текущего дисковода (гл.16).
1A Установка области передачи данных (DTA).
1B Получение таблицы FAT для текущего дисковода.
1C Получение FAT для любого дисковода.
21 Чтение с диска с прямым доступом (гл.16).
22 Запись на диск с прямым доступом (гл.16).
23 Определение размера файла.
24 Установка номера записи для прямого доступа.
25 Установка вектора прерывания.
26 Создание программного сегмента.
27 Чтение блока записей с прямым доступом (гл.16).
28 Запись блока с прямым доступом (гл.16).
29 Преобразование имени файла во внутренние параметры.
2A Получение даты (CX-год,DH-месяц,DL-день).
2B Установка даты.
2C Получение времени (CH-час,CL-мин,DH-с,DL-1/100с).
2D Установка времени.
2E Установка/отмена верификации записи на диск.

Следующие расширенные функции возможны в DOS начиная с версии 2.0:

2F Получение адреса DTA в регистровой паре ES:BX.
30 Получение номера версии DOS в регистре AX.
31 Завершение программы, после которого она остается резидентной в памяти.
33 Проверка Ctrl/Break.
35 Получение вектора прерывания (адреса подпрограммы).
36 Получение размера свободного пространства на диске.
38 Получение государственно зависимых форматов.
39 Создание подкаталога (команда MKDIR).
3A Удаление подкаталога (команда RMDIR).
3B Установка текущего каталога (команда CHDIR).
3C Создание файла без использования FCB (гл.17).
3D Открытие файла без использования FCB (гл.17).
3E Закрытие файла без использования FCB (гл.17).
3F Чтение из файла или ввод с устройства (гл.8,17,19).
40 Запись в файл или вывод на устройство (гл.8,17,19).
41 Удаление файла из каталога (гл.17).
42 Установка позиции для последовательного доступа (гл.17).
43 Изменение атрибутов файла (гл.17).
44 Управление вводом-выводом для различных устройств.
45 Дублирование файлового номера.

46 "Склеивание" дублированных файловых номеров.
47 Получение текущего каталога (гл.17).
48 Выделение памяти из свободного пространства.
49 Освобождений выделенной памяти.
4A Изменение длины блока выделенной памяти.
4B Загрузка/выполнение программы (подпроцесса).
4C Завершение подпроцесса с возвратом управления.
4D Получение кода завершения подпроцесса.
4E Начальный поиск файла по шаблону (гл.17).
4F Поиск следующего файла по шаблону (гл.17).
54 Получение состояния верификации.
56 Переименование файла (гл.17).
57 Получение/установка даты и времени изменения файла.

Следующие расширенные функции возможны в DOS начиная с версии 3.0:

59 Получение расширенного кода ошибки.
5A Создание временного файла.
5B Создание нового файла.
5C Блокирование/разблокирование доступа к файлу.
62 Получение адреса префикса программного сегмента (PSP).

В техническом руководстве по DOS представлены подробные описания каждой функции.

РЕЗИДЕНТНЫЕ ПРОГРАММЫ

Существует ряд распространенных фирменных программ (Prokey, Superkey, Homebase, Sidekick и др.), специально разработанных как резидентные, которые находятся в памяти во время работы других программ. Эти программы можно активизировать нажатием определенных клавиш. Такие программы называются резидентными, и они загружаются в память сразу после загрузки DOS перед выполнением обычных программ.

Для того, чтобы оставить резидентную COM-программу в памяти, необходимо вместо команды RET или INT 20H для выхода использовать команду INT 27H или функцию DOS 31H. Для INT 27H следует передать системе в регистре DX размер программы:

```
MOV     DX,prog-size
INT     27H
```

При выполнении программы инициализации DOS резервирует (выделяет) в старших доступных адресах блок памяти и загружает в него резидентную программу. Это наиболее простая часть создания резидентной программы.

Более сложная часть включает программирование механизма активизации резидентной программы, которая хотя и присоединена к DOS, но не является внутренней программой DOS, как DIR, COPY или CLS. Общим подходом является модификация таблицы векторов прерываний таким образом, чтобы резидентная программа, получала управление при нажатии определенных клавиш или их комбинаций, а все остальные передавала через себя. Резидентная программа обычно (но не обязательно) состоит из следующих частей:

1) секции, переопределяющей адреса в таблице векторов прерываний;
2) процедуры, выполняющейся только один раз при загрузке программы и предназначенной для:

- замены адреса в таблице векторов прерываний на собственный адрес;

- установки размера части программы, которая должна стать резидентной;
- использования прерывания DOS для завершения программы и создания резидентной части по указанному размеру;

3) процедуры, которая остается резидентной и активизируется, например, по вводу с клавиатуры или по сигналам таймера.

Процедура инициализации должна создать необходимые условия для того, чтобы обеспечить работу резидентной программы, а затем - высшая жертва! - позволить себе быть стертой в памяти. В результате память будет распределена следующим образом:

Таблица векторов прерываний
IBMBIO.COM и IBMDOS.COM
COMMAND.COM
Резидентная часть программы - остается в памяти
Инициализирующая часть программы - перекрывается следующей программой
Остальная часть доступной памяти

```
TITLE   RESIDENT (COM) Резидентная программа для очистки
;                  экрана и установки цвета при нажатии
;                  Alt+Left Shift
;-------------------------------------------------------
INTTAB  SEGMENT AT 0H         ;Таблица векторов прерываний:
        ORG     9H*4          ; адрес для Int 9H,
KBADDR  LABEL   DWORD         ; двойное слово
INTTAB  ENDS
;-------------------------------------------------------
ROMAREA SEGMENT AT 400H       ;Область параметров BIOS:
        ORG     17H           ; адрес флага клавиатуры,
KBFLAG  DB      ?             ; состояние Alt + Shift
ROMAREA ENDS
;-------------------------------------------------------
CSEG    SEGMENT PARA          ;Сегмент кода
        ASSUME  CS:CS
        ORG     100H
BEGIN:  JMP     INITZ         ;Выполняется только один раз
KBSAVE  DD      ?             ;Для адреса INT 9 BIOS
;              Очистка экрана и установка цветов:
;              --------------------------------
COLORS  PROC    NEAR          ;Процедура выполняется
        PUSH    AX            ; при нажатии Alt+Left Shift
        PUSH    BX
        PUSH    CX            ;Сохранить регистры
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    DS
        PUSH    ES
        PUSHF
        CALL    KBSAV         ;Обработать прерывание
        ASSUME  DS:ROMAREA
        MOV     AX,ROMAREA    ;Установить DS для
        MOV     DS,AX         ; доступа к состоянию
        MOV     AL,KB  AG     ; Alt+Left Shift
        CMP     AL,00001010B  ;Alt+Left Shift нажаты?
        JNE     EXIT          ; нет - выйти
```

```
      MOV    AX,0600H      ;Функция прокрутки
      MOV    BH,61H        ;Установить цвет
      MOV    CX,00
      MOV    DX,18 FH
      INT    10H
EXIT:
      POP    ES            ;Восстановить регистры
      POP    DS
      POP    DI
      POP    SI
      POP    DX
      POP    CX
      POP    BX
      POP    AX
      IRET                 ;Вернуться
COLORS  ENDP
;           Подпрограмма инициализации:
;           --------------------------
INITZE  PROC    NEAR           ;Выполнять только один раз
      ASSUME  DS:INTTAB
      PUSH   DS              ;Обеспечить возврат в DOS
      MOV    AX,INTTAB       ;Установить сегмент данных
      MOV    DS,AX
      CLI                    ;Запретить прерывания
                             ;Замена адреса обработчика:
      MOV    AX,WORD PTR KBADDR      ;Сохранить адрес
      MOV    WORD PTR KBSAVE,AX      ; BIOS
      MOV    AX,WORD PTR BADDR+2
      MOV    WORD PTR KBSAVE+2,AX
      MOV    WORD PTR KBADDR,OFFSET COLORS ;Заменить
      MOV    WORD PTR KBADDR+2,CS          ; адрес BIOS
      STI                    ;Разрешить прерывания
      MOV    DX,OFFSET INITZE ;Размер программы
      INT    27H             ;Завершить и остаться
INITZE  ENDP                 ; резидентом

CSEG    ENDS
      END    BEGIN
```

Рис.23.2 Резидентная программа

Пример на рис.23.2 иллюстрирует резидентную программу, которая устанавливает цвет экрана при одновременном нажатии клавиш Alt и Left Shift. Основные моменты, представляющие интерес:

Сегмент INTTAB определяет таблицу векторов прерываний, начинающуюся по адресу 0, а точнее - адрес элемента для прерывания 9 (ввод с клавиатуры), названный в программе KBADDR.

Имя ROMAREA определяет сегмент, начинающийся по адресу 400H, и в нем флаговый байт клавиатуры (KBFLAG), который отражает состояние клавиатуры. Бит 3 в этом байте регистрирует нажатие клавиши Alt, а бит 1 - нажатие клавиши Left Shift.

Сегмент CSEG начинает сегмент обычной COM-программы. Первая выполняемая команда JMP INITZE обходит резидентную часть и передает управление в процедуру инициализации (INITZE) в конце программы. Эта процедура устанавливает в регистре DS адрес таблицы векторов прерывания (INTTAB) и передает адрес элемента таблицы для INT 9 (KBADDR) в поле KBSAVE в резидентной процедуре. Следующим шагом в таблице в KBADDR утанавливается адрес резидентной процедуры (COLORS) (первое слово) и содержимое регистра CS (второе слово). Таким образом, KBADDR содержит

теперь два измененных слова: смещение и значение адреса из регистра CS, которые вместе определяют адрес процедуры COLORS в памяти, куда будут направляться теперь все символы, поступающие с клавиатуры. Затем процедура инициализации заносит в регистр DX размер процедуры COLORS (адрес INITZE на один байт больше, чем адрес конца процедуры COLORS) и прекращает работу, используя INT 27H.

Процедура COLORS является резидентной, и она получает управление при нажатии любой клавиши на клавиатуре. Так как это происходит при работе других программ (например, DOS или текстового редактора), то процедура должна сохранить все регистры, которые она использует (а также несколько других на всякий случай). Затем происходит вызов по адресу KBSAVE, т.е. вызов подпрограммы обработки прерывания, после чего процедура проверяет флаг клавиатуры для определения нажатия клавиш Alt и Left Shift. Если эти клавиши были нажаты, то процедура устанавливает необходимые цвета. Завершающие команды включают восстановление всех запомненных вначале регистров (в обратной последовательности) и выход из обработки прерывания по команде IRET.

Поскольку приведенная программа носит иллюстративный характер, ее можно модифицировать или расширить для собственных целей. Некоторые фирменные программы, также изменяющие адрес в векторной таблице для прерывания 9, не разрешают конкурентное использование резидентных программ, аналогичных рассмотренной в данной главе.

ПОРТЫ

Порт представляет собой устройство, которое соединяет процессор с внешним миром. Через порт процессор получает сигналы с устройств ввода и посылает сигналы на устройство вывода. Теоретически процессор может управлять до 65 536 портами, начиная с нулевого порта. Для управления вводом-выводом непосредственно на уровне порта используются команды IN и OUT:

- Команда IN передает данные из входного порта в регистр AL (байт) или в регистр AX (слово). Формат команды:

```
IN        регистр,порт
```

- Команда OUT передает данные в порт из регистра AL (байт) или из регистра AX (слово). Формат команды:

```
OUT       порт,регистр
```

Номер порта можно указывать статически или динамически:

1. Статическое указание порта возможно при непосредственном использовании значения от 0 до 255:

```
Ввод:   IN     AL.порт#       ;Ввод одного байта
Вывод:  OUT    порт#,AX       ;Вывод одного слова
```

2. Динамическое указание порта устанавливается в регистре DX от 0 до 65535. Этот метод удобен для последовательной обработки нескольких портов. Значение в регистре DX в этом случае увеличивается в цикле на 1. Пример ввода байта из порта 60H:

```
MOV     DX,60H  ;Порт 60H (клавиатура)
IN      AL,DX   ;Ввод байта
```

Ниже приведен список некоторых портов (номера в шестнадцатиричном представлении):

21   Регистры маски прерывании.
40...42   Таймер/счетчик
60   Ввод с клавиатуры
61   Звуковой порт (биты 0 и 1)
201   Управление играми
3B0...3BF Монохромный дисплей и параллельный адаптер печати
3D0...3DF Цветной/графический адаптер
3F0...3F7 Дисковый контроллер

Если, например, программа запрашивает ввод с клавиатуры, то она выдает команду прерывания INT 16H. В этом случае система устанавливает связь с BIOS, которая с помощью команды IN вводит байт с порта 60H.

На практике рекомендуется пользоваться прерываниями DOS и BIOS. Однако можно также успешно обойтись без BIOS при работе с портами 21, 40...42, 60 и 201. Листинги BIOS в техническом руководстве по IBM PC содержат различные примеры команд IN и OUT.

ГЕНЕРАЦИЯ ЗВУКА

Компьютер имеет возможность генерировать звук посредством встроенного динамика с постоянным магнитом. Можно выбрать один из двух способов управления динамиком или использовать оба в комбинации: 1) использование бита 1 порта 21 H для активизации микросхемы Intel 8255A-5 (программируемый периферийный интерфейс) (PP1); 2) использование триггера программируемого интервального таймера Intel 8353-5 (PIT). Часовой генератор выдает сигнал с частотой 1,19318 МГц. Таймер PPI управляет триггером 2 через бит 0 порта 61 H.

Программа на рис.23.3 генерирует серию звуковых нот с возрастающей частотой. Значение в поле DURTION формирует продолжительность звучания каждой ноты, а значение в поле TONE определяет частоту звучания. В начале работы программа считывает содержимое порта 61 H и сохраняет полученное значение. Команда CLI сбрасывает флаг прерываний, обеспечивая равномерное звучание. Интервальный таймер генерирует 18,2 такта в секунду, что (при отсутствии CLI) прерывает выполнение программы и вызывает появление звука.

Значение поля TONE определяет частоту звука: большие значения дают низкую частоту, а малые - высокую. После того, как подпрограмма BIOSPKR исполнит очередную ноту, она увеличивает частоту звука посредством сдвига вправо на 1 бит значения в поле TONE (т.е. делит значение поля TONE на 2). Так как уменьшение TONE в данном примере приводит к сокращению продолжительности звучания, то подпрограмма также увеличивает DURTION посредством сдвига его значения на один бит влево, т.е. удваивает это значение.

Программа завершается, когда содержимое поля TONE уменьшится до нуля. Начальные значения в полях DURTION и TONE не имеют принципиального значения. Можно поэкспериментировать с другими значениями, а также испытать выполнение без команды CLI. Можно изменить программу для генерации звуков с уменьшающейся частотой, установив для этого в поле TONE значение 01, а в поле DURTION - некоторое большее значение. В каждом цикле при этом необходимо увеличивать значение в поле TONE и уменьшать значение в поле. DURTION. При достижении нуля в DURTION программу можно завершить.

Можно использовать любые варианты исполнения последовательности нот, например, для привлечения внимания пользователя.

```
TITLE   SOUND   (COM) Процедура для генерации звука
SOUNSG  SEGMENT PARA 'Code'
        ASSUME  CS:SOUNG,DS:SOUNG,SS:SOUNG
```

```
       ORG     100H
BEGIN:  JMP     SHORT MAIN
; -----------------------------------------------------
DURTION DW      1000          ;Время звучания
TONE    DW      256H          ;Высота (частота) звука
; -----------------------------------------------------
MAIN    PROC    NEAR
       IN      AL,61H         ;Получить и сохранить
       PUSH    AX             ; данные порта
       CLI                   ;Запретить прерывания
       CALL    B10SPKR        ;Произвести звук
       POP     AX             ;Восстановить значение
       OUT     61H,AL         ; порта
       STI                   ;Разрешить прерывания
       RET
MAIN    ENDP

B10SPKR PROC    NEAR
B20:    MOV     DX,DURTION     ;Установить время звучания
B30:
       AND     AL,11111100B   ;Очистить биты 0 и 1
       OUT     61H,AL         ;Передать на динамик

       MOV     CX,TONE        ;Установить частоту
B40:
       LOOP    B40            ;Задержка времени
       OR      AL,00000010B   ;Установить бит 1
       OUT     61H,AL         ;Передать на динамик
       MOV     CX,TONE        ;становить частоту
B50:
       LOOP    B50            ;Задержка времени
       DEC     DX             ;Уменьшить время звучания
       JNZ     B30            ;Продолжать?
       SHL     DURTION,1      ; нет - увеличить время,
       SHR     TONE,1         ; сократить частоту
       JNZ     B20            ;Нулевая частота?
       RET                   ; да - выйти
B10SPKR ENDP

SOUNSG  ENDS
       END     BEGIN
```

Рис.23.3 Генерация звука

ГЛАВА 24 Справочник по директивам языка Ассемблер

Ц е л ь: подробно описать операторы и директивы языка Ассемблер.

ВВЕДЕНИЕ

Некоторые особенности Ассемблера кажутся на первый взгляд несколько странными. Но после того, как вы ознакомились с простейшими и наиболее общими свойствами Ассемблера, описанными в предыдущих главах, то обнаружите, что описания в этой главе более понятны и являются удобным руководством к программированию. В данной главе представлены операторы

атрибутов, операторы, возвращающие значение, директивы данных, а также индексная адресация памяти.

ИНДЕКСНАЯ АДРЕСАЦИЯ ПАМЯТИ

При прямой адресации памяти в одном из операндов команды указывается имя определенной переменной, например для переменной COUNTER:

```
ADD  CX,COUNTER
```

Во время выполнения программы процессор локализует указанную переменную в памяти путем объединения величины смещения к этой переменной с адресом сегмента данных.

При индексной адресации памяти ссылка на операнд определяется через базовый или индексный регистр, константы, переменные смещения и простые переменные. Квадратные скобки, определяющие операнды индексной адресации, действуют как знак плюс (+). Для индексном адресации памяти можно использовать:

- базовый регистр BX в виде [BX] вместе с сегментным регистром DS или базовый регистр BP в виде [BP] вместе с сегментным регистром SS. Например, с помощью команды

```
MOV  DX,[BX]        ;Базовый регистр
```

в регистр DX пересылается элемент, взятый по относительному адресу в регистре BX и абсолютному адресу сегмента в регистре DS;

- индексный регистр DI в виде [DI] или индексный регистр SI в виде [SI], оба вместе с сегментным регистром DS. Например, с помощью команды

```
MOV  AX,[SI]        ;Индексный регистр
```

в регистр AX пересылается элемент, взятый по относительному адресу в регистре SI и абсолютному адресу сегмента в регистре DS;

- [константу], содержащую непосредственный номер или имя в квадратных скобках. Например, с помощью команды

```
MOV  [BX+SI+4],AX   ;База+индекс+константа
```

содержимое регистра AX пересылается по адресу, который вычисляется, как сумма абсолютного адреса в регистре DS, относительного адреса в регистре BX, относительного адреса в регистре SI и константы 4;

- смещение (+ или -) совместно с индексным операндом. Существует небольшое различие при использовании константы и смещения. Например, с помощью команды

```
MOV  DX,8[DI][4]    ;Смещение+индекс+константа
```

в регистр DX пересылается элемент, взятый по абсолютному адресу в регистре DS, смещению 8, относительному адресу в регистре DI и константе 4.

Эти операнды можно комбинировать в любой последовательности. Но нельзя использовать одновременно два базовых регистра [BX + BP] или два индексных регистра [DI + SI]. Обычно индексированные адреса используются для локализации элементов данных в таблицах.

ОПЕРАТОРЫ ЯЗЫКА АССЕМБЛЕР

Существует три типа ассемблерных операторов: операторы атрибута, операторы, возвращающие значение, и операторы. специфицирующие битовую строку. В данном разделе рассмотрены первые два типа операторов. Операторы, специфицирующие битовую строку, оператор MASK, счетчик сдвига и оператор WIDTH относятся к директиве RECORD и будет рассмотрены в следующем разделе.

Оператор LENGTH

Оператор LENGTH возвращает число элементов, определенных операндом DUP. Например, следующая команда MOV заносит в регистр DX значение 10:

```
TABLEA    DW     10 DUP(?)
          ...
          MOV    DX,LENGTH TABLEA
```

Если операнд DUP отсутствует, то оператор LENGTH возвращает значение 01. См. операторы SIZE и TYPE в этом разделе.

Оператор OFFSET

Оператор OFFSET возвращает относительный адрес переменной или метки внутри сегмента данных или кода. Оператор имеет следующий формат:

```
OFFSET переменная или метка
```

Например, команда

```
MOV  DX,OFFSET TABLEA
```

устанавливает в регистре DX относительный адрес (смещение) поля TABLEA в сегменте данных. (Заметим, что команда LEA выполняет аналогичное действие, но без использования оператора OFFSET.)

Оператор PTR

Оператор PTR используется совместно с атрибутами типа BYTE, WORD или DWORD для локальной отмены определенных типов (DB, DW или DD) или с атрибутами NEAR или FAR для отмены значения дистанции по умолчанию. Формат оператора следующий:

```
тип  PTR  выражение
```

В поле "тип" указывается новый атрибут, например BYTE. Выражение имеет ссылку на переменную или константу. Приведем несколько примеров оператора PTR:

```
FLDB DB       22H
     DB       35H
FLDW DW       2672H             ;0бьектный код 7226
     MOV      AH,BYTE PTR FLDW     ;Пересылает 1-й байт (72)
     ADD      BL,BYTE PTR FLDW+1   ;Прибавляет 2-й байт (26)
     MOV      BYTE PTR FLDW,05     ;Пересылает 05 в 1-й байт
     MOV      AX,WORD PTR FLDB     ;Заносит в AX байты (2235)
     CALL     FAR PTR[BX]          ;Длинный вызов процедуры
```

Директива LABEL, описанная в следующем разделе, выполняет функцию, аналогичную оператору PTR.

Оператор SEG

Оператор SEG возвращает адрес сегмента, в котором расположена указанная переменная или метка. Наиболее подходящим является использование этого оператора в программах, состоящих из нескольких отдельно ассемблируемых сегментов. Формат оператора:

```
SEG переменная или метка
```

Примеры применения оператора SEG в командах MOV:

```
MOV     DX,SEG FLOW    ;Адрес сегмента данных
MOV     DX,SEG A20     ;Адрес сегмента кода
```

Оператор SHORT

Назначение оператора SHORT - модификация атрибута NEAR в команде JMP, если переход не превышает границы +127 и -128 байт:

```
JMP     SHORT метка
```

В результате ассемблер сокращает машинный код операнда от двух до одного байта. Эта возможность оказывается полезной для коротких переходов вперед, так как в этом случае ассемблер не может сам определить расстояние до адреса перехода и резервирует два байта при отсутствии оператора SHORT.

Оператор SIZE

Оператор SIZE возвращает произведение длины LENGTH и типа TYPE и полезен только при ссылках на переменную с операндом DUP. Формат оператора:

```
SIZE переменная
```

См. пример для оператора TYPE.

Оператор TYPE

Оператор TYPE возвращает число байтов, соответствующее определению указанной переменной:

Определение	Число байтов
DB	1
DW	2
DD	4
DQ	8
DT	10
STRUC	Число байтов, определённых в STRUC
NEAR метка	FFFF
FAR метка	FFFE

Формат оператора TYPE:

TYPE   переменная или метка

Ниже приведены примеры, иллюстрирующие применение операторов TYPE, LENGTH и SIZE:

```
FLDB      DB   ?
TABLEA    DW   20 DUP(?)          ;Определение 20 слов
         ...
         MOV  AX,TYPE FLDB        ;AX = 0001
         MOV  AX,TYPE TABLEA      ;AX = 0002
         MOV  CX,LENGTH TABLEA    ;CX = 000A (10)
         MOV  DX,SIZE TABLEA      ;DX = 0014 (20)
```

Так как область TABLEA определена как DW, то оператор TYPE возвращает 0002H, LENGTH - 000AH (соответственно операнду DUP) и SIZE - произведение типа и длины, т.е. 14H (20).

ДИРЕКТИВЫ АССЕМБЛЕРА

В данном разделе описано большинство ассемблерных директив. В гл.5 были подробно рассмотрены директивы для определения данных (DB, DW и проч.).

Директива ASSUME

Назначение директивы ASSUME - установить для ассемблера связь между сегментами и сегментными регистрами CS, DS, ES и SS. Формат директивы:

```
ASSUME    сегментный_регистр:имя   [, ... ]
```

В директиве указываются имена сегментных регистров, групп (GROUP) и выражений SEG. Одна директива ASSUME может назначить до четырех сегментных регистров в любой последовательности, например:

```
ASSUME    CS:CODESG,DS:DATASG,SS:STACK,ES:DATASG
```

Для отмены любого ранее назначенного в директиве ASSUME сегментного регистра необходимо использовать ключевое слово NOTHING:

```
ASSUME    ES:NOTHING
```

Если, например, регистр DS оказался не назначен или отменен ключевым словом NOTHING, то для ссылки к элементу из сегмента данных в командах используется операнд со ссылкой к регистру DS:

```
MOV       AX,DS:[BX]     ;Использование индексного адреса
MOV       AX,DS:FLDW     ;Пересылка содержимого поля FLDW
```

Конечно, регистр DS должен содержать правильное значение сегментного адреса.

Директива EXTRN

Назначение директивы EXTRN - информировать ассемблер о переменных и метках, которые определены в других модулях, но имеют ссылки из данного модуля. Формат директивы:

```
          EXTRN     имя: тип [, ... ]
```

Директива EXTRN подробно рассмотрена в гл.21.

Директива GROUP

Программа может содержать несколько сегментов одного типа (код, данные, стек). Назначение директивы GROUP - собрать однотипные сегменты под одно имя так, чтобы они поместились в один сегмент объемом 64 Кбайт, формат директивы:

```
          имя     GROUP  имя сегмента  [, ... ]
```

Следующая директива GROUP объединяет SEG1 и SEG2 в одном ассемблерном модуле:

```
          GROUPX       GROUP        SEG1,SEG2
          SEG1         SEGMENT      PARA 'CODE'
          ASSUME         CS:GROUPX
          SEG1         ENDS
          SEG2         SEGMENT      PARA 'CODE'
          ASSUME         CS:GROUPX
          SEG2         ENDS
```

Директива INCLUDE

Отдельные фрагменты ассемблерного кода или макрокоманды могут использоваться в различных программах. Для этого такие фрагменты и макрокоманды записываются в отдельные дисковые файлы, доступные для использования из любых программ. Пусть некоторая подпрограмма, преобразующая ASCII-код в двоичное представление, записана на диске C в файле по имени CONVERT.LIB. Для доступа к этому файлу необходимо указать директиву

```
          INCLUDE C:CONVERT.LIB
```

причем в том месте исходной программы, где должна быть закодирована подпрограмма преобразования ASCII-кода. В результате ассемблер найдет необходимый файл на диске и вставит его содержимое в исходную программу. (Если файл не будет найден, то ассемблер выдаст соответствующее сообщение об ошибке и директива INCLUDE будет игнорирована.)

Для каждой вставленной строки ассемблер выводит в LST-файл в 30-й колонке символ C (исходный текст в LST-файле начинается с 33-й колонки).

В гл.20 (Макрокоманды) дан практический пример директивы INCLUDE и дано объяснение, каким образом можно использовать эту директиву только в первом проходе ассемблера.

Директива LABEL

Директива LABEL позволяет переопределять атрибут определенного имени. Формат директивы:

```
          имя  LABEL     тип
```

В качестве типа можно использовать BYTE, WORD или DWORD для переопределения областей данных или имен структур или битовых строк. Директивой LABEL можно переопределить выполнимый код, как NEAR или FAR. Эта директива позволяет, например, определить некоторое поле и как DB, и

как DW. Ниже проиллюстрировано использование типов BYTE и WORD:

```
REDEFB  LABEL  BYTE
FIELDW  DW     2532H
REDEFW  LABEL  WORD
FIELDB  DB     25H
        DB     32H
        MOV    AL,REDEFB      ;Пересылка первого байта
        MOV    BX,REDEFW      ;Пересылка двух байтов
```

Первая команда MOV пересылает только первый байт поля FIELDW. Вторая команда MOV пересылает два байта, начинающихся по адресу FIELDB. Оператор PTR выполняет аналогичные действия.

Директива NAME

Директива NAME обеспечивает другой способ назначения имени модулю:

```
NAME  имя
```

Ассемблер выбирает имя модуля в следующем порядке:

1) если директива NAME присутствует, то ее операнд становится именем модуля;
2) если директива NAME отсутствует, то ассемблер использует первые шесть символов из директивы TITLE;
3) если обе директивы NAME и TITLE отсутствуют, то именем модуля становится имя исходного файла.

Выбранное имя передается ассемблером в компоновщик.

Директива ORG

Для определения относительной позиции в сегменте данных или кода ассемблер использует адресный счетчик. Рассмотрим сегмент данных со следующими определениями:

Смещение	Имя	Операция	Операнд	Адресный счетчик
00	FLDA	DW	2542H	02
02	FLDB	DB	36H	03
03	FLDC	DW	212EH	05
05	FLDD	DD	00000705H	09

Начальное значение адресного счетчика - 00. Так как поле FLDA занимает два байта, то для получения адреса следующего элемента адресный счетчик увеличивается до значения 02. Поле FLDB занимает один байт, значит значение адресного счетчика увеличивается до значения 03 и т.д.. Для изменения значения адресного счетчика и соответственно адреса следующего определяемого элемента используется директива ORG. Формат директивы:

```
OR6  выражение
```

Выражение может быть абсолютным числом, но не символическим именем, и должно формировать двухбайтовое абсолютное число. Рассмотрим следующие элементы данных, определенные непосредственно после поля FLDD:

Смещение   Имя   Операция   Операнд   Адресный счетчик

```
             ORG        0            00
      00     FLDX   DB         ?            01
      01     FLDY   DW         ?            02
      03     FLDZ   DB         ?            04
             ORG        $+5          09
```

Первая директива ORG возвращает адресный счетчик в нулевое значение. Поля FLDX, FLDY и FLDZ определяют те же области памяти, что и поля FLDA, FLDB и FLDC:

```
      Смещение:  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
                   |       |   |       |
                 FLDA    FLDB FLDC   FLDD
                   |   |       |
                 FLDX FLDY    FLDZ
```

Операнд, содержащий символ доллара ($), как во второй директиве ORG, имеет текущее значение адресного счетчика. Таким образом, операнд $+5 устанавливает адресный счетчик равным 04 + 5 = 09, что представляет собой то же значение, что и после определения поля FLDD.

Метка FLDC указывает на поле длиной в одно слово, находящееся по смещению 03, а метка FLDZ указывает на однобайтовое поле по тому же смещению:

```
        MOV    AX,FLOC        ;Одно слово
        MOV    AL,FLDZ        ;Один байт
```

Директиву ORG можно использовать для переопределения областей памяти. При этом следует правильно устанавливать адресный счетчик и учитывать все переопределяемые адреса памяти. Кроме того, новые переменные не должны определять константы, так как при этом будут перекрыты константы, определенные ранее. Директиву ORG нельзя использовать внутри определения STRUCT.

Директива PROC

Любая процедура представляет собой совокупность кодов, начинающуюся директивой PROC и завершающуюся директивой ENDP. Обычно эти директивы используются для подпрограмм в кодовом сегменте. Ассемблер допускает переход на процедуру с помощью команды JMP, но обычной практикой является использование команды CALL для вызова процедуры и RET для выхода из процедуры.

Процедура, находящаяся в одном сегменте с вызывающей процедурой, имеет тип NEAR:

```
        имя-процедуры  PROC    [NEAR]
```

Если операнд опущен, то ассемблер принимает значение NEAR по умолчанию. Если процедура является внешней по отношению к вызывающему сегменту, то ее вызов может осуществляться только командой CALL, а сама процедура должна быть объявлена как PUBLIC. Более того, если в вызываемой процедуре используется другое значение ASSUME CS, то необходимо кодировать атрибут FAR:

```
                  PUBLIC  имя-процедуры,
        имя-процедуры  PROC    FAR
```

При вызове любой процедуры с помощью команды CALL необходимо обеспечить возврат по команде RET.

Директива PUBLIC

Назначение директивы PUBLIC - информировать ассемблер, что на указанные имена имеются ссылки из других ассемблерных модулей. Формат директивы:

```
PUBLIC имя [,...]
```

Директива PUBLIC подробно описана в гл.21.

Директива RECORD

Директива RECORD позволяет определять битовые строки. Одно из назначений этой директивы - определить однобитовые или многобитовые переключатели. Формат директивы:

```
имя RECORD имя-поля:ширина [=выражение] [, ... ]
```

Имя директивы и имена полей могут быть любыми уникальными идентификаторами. После каждого имени поля следует двоеточие (:) и размер поля в битах, которое может быть от 1 до 16 бит:

Число определенных битов	Принимаемый размер
1...8	8
9...16	16

Любой размер поля до 8 бит представляется восемью битами, а от 9 до 16 бит - представляется шестнадцатью битами, выровненными справа (если необходимо). Рассмотрим следующую директиву RECORD:

```
BITREC RECORD BIT1:3,BIT2:7,BIT3:6
```

Имя BIT1 определяет первые 3 бит поля BITREC, BIT2 - следующие 7 бит и BIT3 - последние 6 бит. Общее число битов - 16, т.е. одно слово. Можно инициализировать поле BITREC, например, следующим образом:

```
BITREC2 RECORD BIT1:3=101B,BIT2:7=0110110B,BIT3:011010B
```

Предположим, что директива RECORD находится перед сегментом данных. Тогда внутри сегмента данных должен быть другой оператор, который отводит память под данные. Для этого необходимо определить уникальное имя, имя директивы RECORD и операнд, состоящий из угловых скобок (символы меньше и больше):

```
DEFBITS BITREC <>
```

Данное определение генерирует объектный код AD9A. который записывается как 9AAD в сегмент данных. В угловых скобках может находиться значение, переопределяющее BITREC.

Программа на рис.24.1 иллюстрирует определение BITREC директивой RECORD, но без начальных значений. В этом случае соответствующий оператор в сегменте данных инициализирует каждое поле операндом в угловых скобках.

Дополнительно к директиве RECORD имеются операторы WIDTH, MASK и фактор сдвига. Использование этих операторов позволяет изменять определение директивы RECORD без изменения команд, которые имеют ссылки на директиву RECORD.

Оператор WIDTH. Оператор WIDTH возвращает число битов в директиве RECORD или в одном из ее полей. На рис.24.1 после метки A10 имеется два примера оператора WIDTH. Первая команда MOV загружает в регистр BH число битов во всем поле RECORD BITREC (16 бит); вторая команда MOV загружает в регистр AL число битов в поле BIT2 (7 бит). В обоих случаях ассемблер генерирует для числа битов непосредственный операнд.

```
                    TITLE   RECORD (COM) Проверка директивы RECORD
0000            CODESG SEGMENT PARA 'Code'
                    ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
0100                  ORG     100H
0100 EB 02      BEGIN:        JMP    SHORT MAIN
              ; -----------------------------------------------------
              BITREC RECORD  BIT1:3,BIT2:7,BIT3:6    ;Определить запись
0102 9A AD      DEFBITS BITREC <101B,0110110B,011010B> ;Инициализировать биты
              ; -----------------------------------------------------
0104            MAIN   PROC    NEAR
0104            A10:                          ;Ширина:
0104 B7 10            MOV     BH,WIDTH BITREC   ; записи (16)
0106 B0 07            MOV     AL,WIDTH BIT2     ; поля (07)
0108            B10:                          ;Величина сдвига:
0108 B1 0D            MOV     CL,BIT1           ; шест.0D
010A B1 06            MOV     CL,BIT2           ;      06
010C B1 00            MOV     CL,BIT3           ;      00
010E            C10:                          ;Маска:
010E B8 E000          MOV     AX,MASK BIT1      ; шест.E000
0111 BB 1FC0          MOV     BX,MASK BIT2      ;      1FC0
0114 B9 003F          MOV     CX,MASK BIT3      ;      003F
0117            D10:                          ;Выделение BIT2:
0117 A1 0102 R        MOV     AX,DEFBITS        ; получить запись,
011A 25 1FC0          AND     AX,MASK BIT2      ; очистить BIT1 и BIT3,
011D B1 06            MOV     CL,BIT2           ; получить сдвиг 06,
011F D3 E8            SHR     AX,CL             ; сдвинуть вправо
0121            E10:                          ;Выделение BIT1:
0121 A1 0102 R        MOV     AX,DEFBITS        ; получить запись,
0124 B1 0D            MOV     CL,BIT1           ; получить сдвиг 13,
0126 D3 E8            SHR     AX,CL             ; сдвинуть вправо
0128 C3               RET
0129                MAIN    ENDP
0129           CODESG  ENDS
                    END     BEGIN
```

```
Structures and records:
           N a m e          Widht  # fields
                              Shift   Widht   Mask    Initial
  BITREC . . . . . . . . . . .  0010   0003
  BIT1 . . . . . . . . . . . .  000D   0003    E000    0000
  BIT2 . . . . . . . . . . . .  0006   0007    1FC0    0000
  BIT3 . . . . . . . . . . . .  0000   0006    003F    0000

Segments and Groups:
           N a m e          Size   Align   Combine Class
CODESG . . . . . . . . . . . .  0129   PARA    NONE    'CODE'

Symbols:
           N a m e          Type    Value   Attr
A10. . . . . . . . . . . . . . L NEAR  0104    CODESG
B10. . . . . . . . . . . . . . L NEAR  0108    CODESG
```

```
BEGIN. . . . . . . . . . . . . L NEAR   0100    CODESG
C10. . . . . . . . . . . . . . L NEAR   010E    CODESG
D10. . . . . . . . . . . . . . L NEAR   0117    CODESG
DEFBITS. . . . . . . . . . . . L WORD   0102    CODESG
E10. . . . . . . . . . . . . . L NEAR   0121    CODESG
MAIN . . . . . . . . . . . . . N PROC   0104    CODESG  Length =0025
```

Рис.24.1. Использование директивы RECORD

Ф а к т о р с д в и г а. Прямая ссылка на элемент в RECORD,например:

```
MOV    CL,BIT2
```

в действительности не имеет отношения к содержимому BIT2. Вместо этого ассемблер генерирует непосредственный операнд, который содержит "фактор сдвига", помогающий изолировать необходимое поле. Непосредственное значение представляет собой число, на которое необходимо сдвинуть BIT2 для выравнивания справа. На рис.24.1 после метки B10 имеются три команды, которые загружают в регистр CL фактор сдвига для полей BIT1, BIT2 и BIT3.

О п е р а т о р MASK. Оператор MASK возвращает "маску" из единичных битовых значений, которые представляют специфицированное поле, иными словами, определяют битовые позиции, которые занимает поле. Например, оператор MASK для каждого из полей, определенных в области BITREC, возвращает следующие значения:

Поле	Двоичное значение	Шестнадцатиричное значение
B1T1	1110000000000000	E000
B1T2	0001111111000000	1FC0
B1T3	0000000000111111	003F

На рис.24.1 три команды после метки C10 загружают в регистры значения оператора MASK для полей BIT1, BIT2 и BIT3. Команды после меток D10 и E10 иллюстрируют выделение значений полей BIT2 и BIT1 соответственно из области BITREC. После метки D10 в регистр AX загружается все значение области, определенной директивой RECORD, а затем из этого значения с помощью оператора MASK выделяются только биты поля BIT2:

```
Область RECORD:     101 0110110 011010
AND MASK BIT2:      000 1111111 000000
Результат:          000 0110110 000000
```

В результате сбрасываются все биты, кроме принадлежащих к полю BIT2. Следующие две команды приводят к сдвигу содержимого регистра AX на шесть битов для выравнивания справа:

```
0000000000110110      (0036H)
```

После метки E10 в регистр AX загружается все значение области, определенной директивой RECORD, и так как BIT1 является самым левым полем, то в примере используется только фактор для сдвига значения вправо на 13 бит:

```
0000000000000101      (0005H)
```

Директива SEGMENT

Ассемблерный модуль может состоять из одного или более сегментов, части сегмента или даже частей нескольких сегментов. Формат директивы:

```
имя_сегмента SEGMENT [выравнивание] [объединение] [класс]
              .
              .
              .
имя_сегмента   ENDS
```

Все операнды являются необязательными. Ниже описаны операнды для выравнивания, объединения и указания класса.

В ы р а в н и в а н и е. Операнд выравнивания определяет начальную границу сегмента, например

```
PAGE   = xxx00
PARA   = xxxx0 (граница по умолчанию)
WORD   = xxxxe (четная граница)
BYTE   = xxxxx
```

где x - любая шестнадцатиричная цифра,
e - четная шестнадцатиричная цифра.

О б ъ е д и н е н и е. Операнд объединения указывает способ обработки сегмента, при компоновке:

NONE: Значение по умолчанию. Сегмент должен быть логически отделен от других сегментов, хотя физически он может быть смежным. Предполагается, что сегмент имеет собственный базовый адрес;

PUBLIC: Все PUBLIC - сегменты, имеющие одинаковое имя и класс, загружаются компоновщиком в смежные области. Все такие сегменты имеют один общий базовый адрес;

STACK: Для компоновщика операнд STACK аналогичен операнду PUBLIC. В любой компонуемой программе должен быть определен по крайней мере один сегмент STACK. Если объявлено более одного стека, то стековый указатель (SP) устанавливается на начало первого стека;

COMMON: Для сегментов COMMON с одинаковыми именами и классами компоновщик устанавливает один общий базовый адрес. При выполнении происходит наложение второго сегмента на первый. Размер общей области определяется самым длинным сегментом;

AT-параграф: Параграф должен быть определен предварительно. Данный операнд обеспечивает определение меток и переменных по фиксированным адресам в фиксированных областях памяти, таких, как ROM или таблица векторов прерываний в младших адресах памяти. Например, для определения адреса дисплейного видеобуфера используется

```
VIDEO_RAM      SEGMENT AT 0B800H
```

Класс: Операнд класс может содержать любое правильное имя, заключенное в одиночные кавычки. Данный операнд используется компоновщиком для обработки сегментов, имеющих одинаковые имена и классы. Типичными примерами являются классы 'STACK' и 'CODE'.

Следующие два сегмента объединяются компоновщиком в один физический сегмент при одном значении сегментного регистра:

```
Ассемблерный   SEG1 SEGMENT   PARA PUBLIC 'CODE'
модуль 1             ASSUME  CS:SEG1
                  ...
               SEG1   ENDS
               -------------------------------
Ассемблерный   SEG2 SEGMENT   PARA PUBLIC 'CODE'
модуль 2             ASSUME  CS:SEG1
                  ...
               SEG2   ENDS
               -------------------------------
```

Сегменты могут быть вложенными один в другой:

```
SEG1   SEGMENT
    ...             Начало SEG1
SEG2   SEGMENT
    ...             Область SEG2
SEG2   ENDS
    ...             Конец SEG1
SEG1   ENDS
```

Для объединения сегментов в группы используйте директиву GROUP.

Директива STRUC

Директива STRUC обеспечивает определение различных полей в виде структуры. Данная директива не поддерживается в малом ассемблере ASM. Формат директивы:

```
Имя-структуры  STRUC
               ...
               [определение полей данных]
               ...
Имя-структуры  ENDS
```

Структура начинается собственным именем в директиве STRUC и завершается таким же именем в директиве ENDS. Ассемблер записывает поля; определенные в структуре, одно за другим от начала структуры. Правильными операторами определения полей являются DB, DW, DD и DT с указанием имен или без них.

На рис.24.2 директива STRUC определяет список параметров PARLIST для ввода имени с клавиатуры. Следующий далее оператор выделяет память под данную структуру:

```
PARAMS    PARLIST <>
```

Данный оператор обеспечивает адресацию структуры внутри программы. Угловые скобки (символы меньше и больше) в данном случае пусты, но в них можно указать данные для переопределения областей внутри структуры.

В командах ассемблера может использоваться прямая адресация по имени структуры. Для ссылки на определенное поле внутри структуры в командах используется имя структуры (PARAMS в данном примере) и через точку имя конкретного поля:

```
MOV       AL,PARAMS.ACTLEN
```

Используя оператор выделения памяти, можно переопределить содержимое полей внутри структуры. Правила для практического использования этой возможности можно найти в руководстве по Ассемблеру.

```
                    TITLE   DSTRUC  (COM) Определение структуры
0000                  CODESG  SEGMENT PARA 'Code'
                          ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
0100                        ORG     100H
0100 EB 29             BEGIN:  JMP     SHORT MAIN
                    ; -----------------------------------------
                    PARLIST        STRUC          ;Список параметров
0000 19                MAXLEN  DB      25              ;
0001 ??               ACTLEN  DB      ?              ;
0002    19 [ 20 ]       NAMEIN  DB      25 DUP(' ')     ;
001B                  PARLIST        ENDS
                                                ;
0102 19                PARAMS  PARLIST <>              ;Область структуры
0103 ??
0104    19 [ 20 ]
011D 57 68 61 74 20 69  PROMPT  DB      'What is name?', 'S'
    73 20 6E 61 6D 65
    3F 24
                    ; -----------------------------------------
012B                  MAIN    PROC    NEAR
012B B4 09                     MOV     AH,09           ;Выдать запрос
012D 8D 16 011D R                  LEA     DX,PROMPT
0131 CD 21                     INT     21H
0133 B4 0A                     MOV     AH,0AH          ;Получить ввод
0135 8D 16 0102 R                 LEA     DX,PARAMS
0139 CD 21                     INT     21H
013B A0 0103 R                   MOV     AL,PARAMS.ACTLEN ;Длина ввода
                    ;       ...
013E C3                       RET
013F                  MAIN    ENDP
013F                  CODESG  ENDS
                          END     BEGIN


Structures and records:
             N a m e         Width  # fields
                             Shift  Width   Masc    Initial
PARLIST. . . . . . . . . . . . . 001B   0003
  MAXLEN . . . . . . . . . . . . 0000
  ACTLEN . . . . . . . . . . . . 0001
  NAMEIN . . . . . . . . . . . . 0002

Segments and Groups:
             N a m e         Size   Align   Combine Class
CODESG . . . . . . . . . . . . . 013F   PARA    NONE    'CODE'

Symbols:
             N a m e         Type   Value   Attr
BEGIN. . . . . . . . . . . . . L NEAR   0100    CODESG
MAIN . . . . . . . . . . . . . N PROC   012B    CODESG  Length =0014
PARAMS . . . . . . . . . . . . L 001B   0102    CODESG
PROMPT . . . . . . . . . . . . L BYTE   011D    CODESG
```

Рис.24.2. Пример определения структуры

ГЛАВА 25   Справочник по командам языка Ассемблер

Ц е л ь: описать набор команд Ассемблера и объяснить их машинные коды.

ВВЕДЕНИЕ

В данной главе приведены объяснения машинных кодов и перечислены в алфавитном порядке символические коды команд с указанием их назначений.

Многие специфические команды имеют однобайтовые машинные коды, например:

```
Объектный код:      Символические команды:
     40            INC  AX       ;Увеличение AX на 1
     50            PUSH AX       ;Запись AX в стек
     C3            RET  (short)  ;Короткий возврат из процедуры
     CB            RET  (far)    ;Длинный возврат из процедуры
     FD            STD           ;Остановка флага направления
```

Ни одна из перечисленных команд не использует прямой адресации памяти. Другие команды, использующие непосредственный операнд, 8-битовый регистр, регистровую пару или адрес памяти, требуют более сложного машинного кода.

ОБОЗНАЧЕНИЕ РЕГИСТРОВ

Команды, использующие регистр, могут содержать три бита, указывающих на конкретный регистр, и один бит "w", определяющий размер регистра: байт или слово. Кроме того, лишь некоторые команды обеспечивают доступ к сегментным регистрам. На рис.25.1 показана полная идентификация регистров.

Рассмотрим команду MOV с однобайтовым непосредственным операндом:

```
MOV    AH,00   10110 100 00000000
                   |   |
                   w  reg = AH
```

Основные, базовые и индексные регистры:

Биты:	w = 0	w = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Биты:	Сегментный регистр:
00	ES
01	CS
10	SS
11	DS

Рис.25.1. Обозначение регистров

В данном случае первый байт машинного кода указывает на однобайтовый размер (w = 0) и на регистр AH (100). Следующая команда MOV содержит непосредственный двухбайтовый операнд:

```
     MOV    AX,00  10111 000 00000000 00000000
                       |   |
                       w  reg = AX
```

Первый байт машинного кода указывает на размер в одно слово (w=1) и на регистр AX (000). Не следует обобщать приведенные примеры, так как указание регистра и бита w может быть в различных позициях кода.

БАЙТ СПОСОБА АДРЕСАЦИИ

Байт способа адресации, если он присутствует, занимает второй байт машинного кода и состоит из следующих трех элементов:

1) mod - двухбитового кода, имеющего значения 11 для ссылки на регистр и 00, 01 и 10 для ссылки на память;
2) reg - трехбитового указателя регистра;
3) r/m - трехбитового указателя регистра или памяти (r - регистр, m - адрес памяти).

Кроме того, первый байт машинного кода может содержать бит "a", который указывает направление потока между операндом 1 и операндом 2.
Рассмотрим пример сложения содержимого регистра AX с содержимым регистра BX:

```
     ADD    BX,AX   00000011  11 011 000
                          dw mod reg r/m
```

В этом примере d=1 означает, что mod (11) и reg (011) описывают операнд 1, а r/m (000) описывает операнд 2. Так как бит w=1, то размер равен одному слову. Таким образом, команда должна прибавить AX (OOQ) к BX (011).
Второй байт команды в объектном коде указывает большинство способов адресации памяти. В следующем разделе способы адресации будут подробно рассмотрены.

Биты MOD

Два бита mod определяют адресацию регистра или памяти. Ниже поясняется их назначение:

00 биты r/m дают абсолютный адрес, байт смещения (относительный адрес) отсутствует;
01 биты r/m дают абсолютный адрес памяти и имеется один байт смещения;
10 биты r/m дают абсолютный адрес и имеется два байта смещения;
11 биты r/m определяют регистр. Бит w (в байте кода операции) определяет ссылку на восьми- или шестнадцатибитовый регистр.

Биты REG

Три бита reg (вместе с битом w) определяют конкретный восьми- или шестнадцатибитовый регистр.

Биты R/M

Три бита r/m (регистр/память) совместно с битами mod определяют способ адресации, как показано на рис.25.2.

r/m	mod=00	mod=01	mod=10	mod=1.1 w=0	mod=11 w=1
000	BX+SI	BX+SI+disp	BX+SI+disp	AL	AX
001	BX+DI	BX+DI+disp	BX+DI+disp	CL	CX
010	BP+SI	BP+SI+disp	BP+SI+disp	DL	DX
011	BP+DI	BP+DI+disp	BP+DI+disp	BL	BX
100	SI	SI+disp	SI+disp	AH	SP
101	DI	DI+disp	DI+disp	CH	BP
110	Direct	BP+disp	BP+disp	DH	SI
111	BX	BX+disp	BX+disp	BH	DI

Рис.25.2. Биты r/m

ДВУХБАЙТОВЫЕ КОМАНДЫ

Рассмотрим пример сложения содержимого регистров BX и AX:

```
ADD    BX,AX    0000 0011  11 011 000
                dw mod reg r/m
```

d 1 означает, что биты reg и w описывают операнд 1 (BX), а биты mod, r/m и w - Операнд 2 (AX);
w 1 определяет размер регистров в одно слово,
mod 11 указывает, что операнд 2 является регистром;
reg 011 указывает, что операнд 1 является регистром BX;
r/m 000 указывает, что операнд 2 является регистром AX.

Рассмотрим пример умножения регистра AL на регистр BL:

```
MUL    BL       11110110  11 100 011
                  w mod reg r/m
```

Команда MUL предполагает, что регистр AL содержит множимое. Размер регистра равен одному байту (w = 0), mod указывает на регистровую операцию, r/m = 011 указывает на регистр BL. В данном случае reg = 100 не имеет смысла.

ТРЕХБАЙТОВЫЕ КОМАНДЫ

Следующая команда MOV генерирует три байте машинного кода:

```
MOV     mem,AX  10100001 dddddddd dddddddd
```

Для команды пересылки из регистра AX или AL необходимо знать, сколько байтов участвует в операции: один или два. В данном примере w = 1 означает

слово, следовательно, предполагается 16-битовый регистр AX. Использование во втором операнде регистра AL приведет к значению бита w = 0. Байты 2 и 3 содержат относительный адрес памяти. Команды, использующие регистры AX или AL, часто генерируют более эффективный (короткий) машинный код.

ЧЕТЫРЕХБАЙТОВЫЕ КОМАНДЫ

Рассмотрим пример умножения регистра AL на значение в памяти. Процессор предполагает, что множимое находится в регистре AL для однобайтового умножения и в регистре AX для Двухбайтового умножения:

```
MUL    mem_byte        11110110  00 100 110
                              w mod reg r/m
```

Для данной команды reg всегда имеет значение 100, mod = 00 указывает на операцию с памятью, а r/m=110 - на прямой способ адресации. Машинная команда также содержит два следующих байта, определяющих относительный адрес памяти.

Рассмотрим еще один пример, иллюстрирующий команду LEA, которая всегда специфицирует двухбайтовый адрес:

```
LEA     DX,mem 10001101  00 010 110
                   LEA   mod reg r/m
```

Reg =010 означает регистр DX. Mod =00 и r/m=110 определяют прямой способ адресации памяти. В следующих двух байтах содержится относительный адрес.

КОМАНДЫ В АЛФАВИТНОМ ПОРЯДКЕ

В данном разделе представлен набор команд Ассемблера в алфавитном порядке. Некоторые команды, например сдвиг и циклический сдвиг, для краткости сгруппированы. Ряд специальных команд для процессоров 80186, 80286 и 80386 выходят за рамки данной книги и поэтому в данной главе также отсутствуют. При пояснении команд и способов адресации используются следующие сокращения:

addr — адрес памяти;
addr-high — первый байт адреса (старший);
addr-low — левый (младший) байт. адреса;
data — непосредственный операнд (8 бит при w=0 и 16 бит при w= 1);
data-high — правый (старший) байт непосредственного операнда;
data-low — левый (младший) байт непосредственного операнда;
disp — смещение (относительный адрес);
reg — ссылка на регистр.

AAA: Коррекция ASCII-формата для сложения

О п е р а ц и я: Корректирует сумму двух ASCII-байтов в регистре AL. Если правые четыре бита регистра AL имеют значение больше 9 или флаг AF установлен в 1, то команда AAA прибавляет к регистру AH единицу и устанавливает флаги AF и CF. Команда всегда очищает четыре левых бита в регистре AL.

Ф л а г и: Команда воздействует на флаги AF и CF (флаги OF, PF, SF и ZF не определены).

О б ъ е к т н ы й к о д: 00110111 (без операндов).

AAD: Коррекция ASCII-формата для деления

О п е р а ц и я: Корректирует ASCII-величины для деления. Команда AAD используется перед делением неупакованных десятичных чисел в регистре AX (удаляет тройки ASCII-кода). Эта команда корректирует делимое в двоичное значение в регистре AL для последующего двоичного деления. Затем умножает содержимое регистра AH на 10. прибавляет результат к содержимому регистра AL и очищает AH. Команда AAD не имеет операндов.

Ф л а г и: Команда воздействует на флаги PF, CF, ZF (флаги AF CF и OF не определены).

О б ъ е к т н ы й к о д: |11010101|00001010|.

AAM: Коррекция ASCII-формата для умножения

О п е р а ц и я: Команда AAM используется для коррекции результата умножения двух неупакованных десятичных чисел. Команда делит содержимое регистра AL на 10, записывает частное в регистр AH, а остаток в регистр AL.

Ф л а г и: Команда воздействует на флаги PF, SF и ZF (флаги AF CF и OF не определены).

О б ъ е к т н ы й к о д: |11010100|00001010| (без операндов).

AAS: Коррекция ASCII-формата для вычитания

О п е р а ц и я: Корректирует разность двух ASCII-байтов в регистре AL. Если первые четыре бита имеют значение больше 9 или флаг CF установлен в 1, то команда AAS вычитает 6 из регистра AL и 1 из регистра AH, флаги AF и CF при этом устанавливаются в 1. Команда всегда очищает левые четыре бита в регистре AL.

Ф л а г и: Команда воздействует на флаги AF и CF (флаги OF PF SF и ZF не определены).

О б ъ е к т н ы й к о д: 00111111 (без операндов).

ADC: Сложение с переносом

О п е р а ц и я: Обычно используется при сложении многословных величин для учета бита переполнения в последующих фазах операции. Если флаг CF установлен в 1, то команда ADC сначала прибавляет 1 к операнду 1. Команда всегда прибавляет операнд 2 к операнду 1, аналогично команде ADD.

Ф л а г и: Команда воздействует на флаги AF, CF, OF, PF, SF и ZF.

О б ъ е к т н ы й к о д (три формата):

Регистр плюс регистр или память:
|000100dw|modregr/m|
Регистр AX (AL) плюс непосредственное значение:
|0001010w|-- data--|data, если w=1|

Регистр или память плюс непосредственное значение:
y100000sw|mod010r/m|--data--|data, если sw=01|

ADD: Сложение двоичных чисел

О п е р а ц и я: Прибавляет один байт или одно слово в памяти, регистре или непосредственно к содержимому регистра или прибавляет один байт или слово в регистре или непосредственно к памяти.

Ф л а г и: Команда воздействует на флаги AF, CF, OF, PF, SF и ZF.

О б ъ е к т н ы й к о д (три формата):

Регистр плюс регистр или память:
|000000dw|modregr/rm|
Регистр AX (AL) плюс непосредственное значение:
|0000010w|--data--|data, если w=1|
Регистр или память плюс непосредственное значение:
|100000sw|mod000r/m|--data--|data, если BW=01|

AND: Логическое И

О п е р а ц и я: Команда выполняет поразрядную конъюнкцию (И) битов двух операндов. Операнды представляют собой одно- или двухбайтовые величины в регистре или памяти. Второй операнд может содержать непосредственные данные. Команда AND проверяет два операнда поразрядно. Если два проверяемых бита равны 1, то в первом операнде устанавливается единичное значение бита, в других случаях - нулевое. См. также команды OR, XOR и TEST.

Ф л а г и: Команда воздействует на флаги CF, OF, PF, SF и ZF (флаг AF не определен).

О б ъ е к т н ы й к о д (три формата):

Регистр И регистр или память:
|001000dw|modregr/m|
Регистр AX (AL) И непосредственное значение:
|0010010w|--data--|data, если w=1|
Регистр или память И непосредственное значение:
|1010000w|modl00r/m|--data--|data, если w=1|

CALL: Вызов процедуры

О п е р а ц и я: Выполняет короткий или длинный вызов процедуры для связи подпрограмм. Для возврата из процедуры используется команда RET. Команда CALL уменьшает содержимое SP на 2 и заносит в стек адрес следующей команды (из IP), а затем устанавливает в регистре IP относительный адрес процедуры. Впоследствии команда RET использует значение в стеке для возврата. Существует четыре типа команды CALL для вызова внутри сегмента и между сегментами. Команда межсегментного вызова сначала уменьшает SP, заносит в стек адрес из регистра CS, а затем загружает в стек внутрисегментный указатель.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д (четыре формата):

Прямая адресация в сегменте:
|11101000|disp-low|diep-high|
Косвенная адресация в сегменте:
|11111111|mod010r/m|
Косвенная адресация между сегментами:
|11111111|mod011r/m|
Прямая адресация между сегментами:
|10011010|offset-low|offset-high|seg-low|seg-high|

CBW: Преобразование байта в слово

О п е р а ц и я: Расширяет однобайтовое арифметическое значение в регистре AL до размеров слова. Команда CBW размножает знаковый бит (7) в регистре AL по всем Битам регистра AH. См также команду CWD.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 10011000 (без операндов).

CLC: Сброс флага переноса

О п е р а ц и я: Устанавливает значение флага переноса, равное 0, так что, например, команда ADC не прибавляет единичный бит. Команда CLC не имеет операндов. См. также команду STD.

Ф л а г и: Команда воздействует на флаг CF (устанавливается в 0).

О б ъ е к т н ы й к о д: 11111000.

CLD: Сброс флага направления

О п е р а ц и я: Устанавливает значение флага направления, равное 0. В результате такие строковые операции, как CMPS или MOVS обрабатывают данные слева направо. См. также команду STD.

Ф л а г и: Команда воздействует на флаг DF (устанавливается в 0).

О б ъ е к т н ы й к о д: 11111100 (без операндов).

CLI: Сброс флага прерывания

О п е р а ц и я: Запрещает маскируемые внешние прерывания по процессорной шине INTR посредством установки значения флага прерывания IF в 0. См. также команду STI.

Ф л а г и: Команда воздействует на флаг IF (устанавливается в 0).

О б ъ е к т н ы й к о д: 11111010 (без операндов).

CMC: Переключение флага переноса

О п е р а ц и я: Инвертирует флаг CF, т.е. преобразует нулевое значение флага CF в единичное и наоборот.

Ф л а г и: Команда воздействует на флаг CF (инвертируется).

О б ъ е к т н ы й к о д: 11110101 (без операндов).

CMP: Сравнение

О п е р а ц и я: Сравнивает содержимое двух полей данных. Фактически команда CMP вычитает второй операнд из первого, но содержимое полей не изменяет. Операнды должны иметь одинаковую длину: байт или слово. Команда CMP может сравнивать содержимое регистра, памяти или непосредственное значение с содержимым регистра; или содержимое регистра или непосредственное значение с содержимым памяти. См. также команду CMPS.

Ф л а г и: Команда воздействует на флаги AF, CF, OF, PF, SF и ZF.

О б ъ е к т н ы й к о д (три формата):

Регистр или память с регистром:
|001110dw|modregr/m|
Непосредственное значение с регистром AX (AL):
|0011110w|--data--|data, если w=1|
Непосредственное значение с регистром или памятью:
|100000sw|mod111r/m|--data--|data, если sw=0|

CMPS/CMPSB/CMPSW: Сравнение строк

О п е р а ц и я: Сравнивают строки любой длины. Этим командам обычно предшествует префикс REPn, например REPE CMPSB. Команда CMPSB сравнивает память по байтам, а команда CMPSW - по словам. Первый операнд этих команд адресуется регистровой парой DS:SI, а второй - регистровой парой ES:DI. Если флаг DF установлен в 0, то сравнение происходит слева направо, регистры SI и DI при этом увеличиваются после каждого сравнения. Если флаг DF установлен в 1, то сравнение происходит справа налево, а регистры SI и DI при этом уменьшаются после каждого сравнения.

Ф л а г и: Команда воздействует на флаги AF, CF, OF, PF, SF и ZF.

О б ъ е к т н ы й к о д: 1010011w.

CWD: Преобразование слова в двойное слово

О п е р а ц и я: Расширяет арифметическое значение в регистре AX до размеров двойного слова в регистровой паре DX:AX, дублируя при этом знаковый бит (15-й бит в регистре AX) через регистр DX. Обычно используется для получения 32-битового делимого. См. также команду CBW.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 10011001 (без операндов).

DAA: Десятичная коррекция для сложения

О п е р а ц и я: Корректирует результат сложения двух BCD (десятичных упакованных) элементов в регистре AL. Если четыре правых бита имеют значение больше 9 или флаг AF установлен в 1, то команда DAA прибавляет 6 к регистру AL и устанавливает флаг AF. Если регистр AL содержит значение больше, чем 9F, или флаг CF установлен в 1, то команда DAA прибавляет 60H к регистру AL и устанавливает флаг CF. См. также команду DAS.

Ф л а г и: Команда воздействует на флаги AF. CF, PF, SF и 2F (флаг OF неопределен).

О б ъ е к т н ы й к о д: 00100111 (без операндов).

DAS: Десятичная коррекция для вычитания

О п е р а ц и я: Корректирует результат вычитания двух BCD (десятичных упакованных) чисел в регистре AL. Если четыре правых бита имеют значение больше 9 или флаг AF установлен в 1, то команда DAS вычитает 60H из регистра AL и устанавливает флаг CF. См. также команду DAA.

Ф л а г и: Команда воздействует на флаги AF, CF, PF, SF и ZF.

О б ъ е к т н ы й к о д: 00101111 (без операндов).

DEC: Декремент

О п е р а ц и я: Вычитает 1 из байта или слова в регистре или в памяти например DEC CX. См. также команду INC.

Ф л а г и: Команда воздействует на флаги AF, OF, PF, SF и ZF.

О б ъ е к т н ы й к о д (два формата):

```
Регистр:            |01001reg|
Регистр или память: |1111111w|mod001r/m|
```

DIV: Деление

О п е р а ц и я: Выполняет деление беззнакового делимого (16 или 32 бит) на беззнаковый делитель (8 или 16 бит). Левый единичный бит рассматривается как бит данных, а не как минус для отрицательных чисел. Для 16-битового деления делимое должно находиться в регистре AX, а 8-битовый делитель возможен в регистре или в памяти, например DIV BH. Частное от деления получается в регистре AL, а остаток - в регистре AH. Для 32-битового деления делимое должно находиться в регистровой паре DX:AX а 16-битовый делитель возможен в регистре или в памяти, например DIV CX. Частное от деления получается в регистре AX, а остаток - в регистре DX. См. также команду IDIV.

Ф л а г и: Команда воздействует на флаги AF, CF, OF, PF SF и ZF (все не определены).

О б ъ е к т н ы й к о д: |1111011w|mod110r/m|

ESC: Переключение на сопроцессор

О п е р а ц и я: Обеспечивает использование сопроцессора для выполнения специальных операций. Для выполнения арифметических операций над числами с плавающей запятой используются сопроцессоры 8087 или 80287. Команда ESC передает в сопроцессор инструкцию и операнд для выполнения необходимой операции.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |11011xxx|modxxxr/m| (x - любое знач.).

HLT: Останов микропроцессора

О п е р а ц и я: Приводит процессор в состояние останова, в котором происходит ожидание прерывания. При завершении команды HLT регистры CS:IP указывают на следующую команду. При возникновении прерывания процессор записывает в стек регистры CS и IP и выполняет подпрограмму обработки прерывания. При возврате из подпрограммы команда IRET восстанавливает регистры CS и IP из стека и управление передается на команду, следующую за командой HLT.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 11110100 (без операндов).

IDIV: Целое деление знаковых величин

О п е р а ц и я: Выполняет деление знакового делимого (16 или 32 бит) на знаковый делитель (8 или 16 бит). Левый единичный бит рассматривается как знак минус для отрицательных чисел. Для 16-битового деления делимое должно находиться в регистре AX, а 8-битовый делитель возможен в регистре или в памяти, например IDIV DL. Частное от деления получается в регистре AL, а остаток - в регистре AH. Для 32-битового деления делимое должно находиться в регистровой паре DX:AX, а 16-битовый делитель возможен в регистре или в памяти, например IDIV BX. Частное от деления получается в регистре AX, а остаток - в регистре DX. См. команды CBW и CWD для расширения знакового делимого, а также команду DIV.

Ф л а г и: Команда воздействует на флаги AF, CF, OF, PF, SF и ZF.

О б ъ е к т н ы й к о д: |1111011w|mod111r/m|

IMUL: Целое умножение знаковых величин

О п е р а ц и я: Выполняет умножение на знаковый множитель (8 или 16 бит). Левый единичный бит рассматривается как знак минус для отрицательных чисел. Для 8-битового умножения множимое должно находиться в регистре AL, а множитель возможен в регистре или в памяти, например IMUL BL. Произведение получается в регистре AX. Для 16-битового умножения множимое должно находиться в регистре AX, а множитель возможен в регистре или в памяти, например IMUL BX. Произведение получается в регистровой паре DX:AX. См. также команду MUL.

Ф л а г и: Команда воздействует на флаги CF и OF (флаги AF PF SF и ZF не определены).

О б ъ е к т н ы й к о д: |1111011w|mod101r/m|

IN: Ввод байта или слова из порта

О п е р а ц и я: Передает из вводного порта один байт в регистр AL или два байта в регистр AX). Порт кодируется как фиксированный числовой операнд (IN AX,порт#) или как переменная в регистре DX (IN AX,DX). Процессоры 80186, 80286 и 80386 имеют, кроме того, команду INS (Input String - ввод строки). См. также команду OUT.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д (два формата):

Переменный порт: |1110110w|
Фиксированный порт: |1110010w|--port--|

INC: Инкремент

О п е р а ц и я: Прибавляет 1 к байту или слову в регистре или в памяти, например INC CX. См. также команду DEC.

Ф л а г и: Команда воздействует на флаги AF, OF, PF, SF и ZF.

О б ъ е к т н ы й к о д (два формата):

Регистр: |01000reg|
Регистр или память: |1111111w|mod000r/m|

INT: Прерывание

О п е р а ц и я: Прерывает выполнение программы и передает управление по одному из 256 адресов (векторов прерывания). Команда INT выполняет следующее: 1) уменьшает значение SP на 2 и заносит в стек флаговый регистр, сбрасывает флаги IF и TF; 2) уменьшает значение SP на 2 и заносит регистр CS в стек, старшее слово из вектора прерывания помещает в регистр CS; 3) уменьшает значение SP на 2 и заносит регистр IP в стек, младшее слово из вектора прерывания помещает в регистр IP.

Ф л а г и: Команда воздействует на флаги IF и TF.

О б ъ е к т н ы й к о д:
|1100110v|--type--| (если v = 0, то type = 3).

INTO: Прерывание по переполнению

О п е р а ц и я: Приводит к прерыванию при возникновении переполнения (флаг OF установлен в 1) и выполняет команду IRET 4. Адрес подпрограммы обработки прерывания (вектор прерывания) находится по адресу 10H. См. также команду INT.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 11001110 (без операндов).

IRET: Возврат из обработки прерывания

О п е р а ц и я: Обеспечивает возврат из подпрограммы обработки прерывания. Команда IRET выполняет следующее: 1) помещает слово из вершины стека в регистр IP и увеличивает значение SP на 2; 2) помещает слово из вершины стека в регистр CS и увеличивает значение SP на 2; 3) помещает слово из вершины стека во флаговый регистр и увеличивает значение SP. на 2. См. также команду RET.

Ф л а г и: Команда воздействует на все флаги.

О б ъ е к т н ы й к о д: 11001111 (бег операндов).

JA/JNBE: Переход по "выше" или "не ниже или равно"

О п е р а ц и я: Используется после проверки беззнаковых данных для передачи управления по другому адресу. Если флаг CF равен нулю (нет переноса) и флаг ZF равен нулю (не нуль), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01110111|--disp--|

JAE/JNB: Переход по "выше или равно" или "не ниже"

О п е р а ц и я: Используется после проверки беззнаковых данных для передачи управления по другому адресу. Если флаг CF равен нулю (нет переноса), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01110011|--disp--|

JB/JNAE: Переход по "ниже" или "не выше или равно"

О п е р а ц и я: Используется после проверки беззнаковых данных для Передачи управления по другому адресу. Если флаг CF равен единице (есть перенос), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01110010|--disp--|

JBE/JNA: Переход по "ниже или равно" или "не выше"

О п е р а ц и я: Используется после проверки беззнаковых данных для передачи управления по другому адресу. Если флаг CF равен единице (есть перенос) или флаг AF равен единице, то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01110110|--disp--|

JC: Переход по переносу

О п е р а ц и я: Идентична JB/JNAE.

JCXZ: Переход по "CX равно нулю"

О п е р а ц и я: Выполняет передачу управления по указанному в операнде адресу, если значение в регистре CX равно нулю. Команда JCXZ может быть полезна в начале циклов LOOP.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |11100011|--disp--|

JE/JZ: Переход по "равно" или по "нулю"

О п е р а ц и я: Используется после проверки знаковых или беззнаковых данных для передачи управления по другому адресу. Если флаг ZF равен единице (нулевое состояние), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01110100|--disp--|

JG/JNLE: Переход по "больше" или "не меньше или равно"

О п е р а ц и я: Используется после проверки знаковых данных для передачи управления по другому адресу. Если флаг ZF равен нулю (не нуль) и флаги SF и OF одинаковы (оба 0 или оба 1), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01111111|--disp--|

JGE/JNL: Переход по "больше или равно" или "не меньше"

О п е р а ц и я: Используется после проверки знаковых данных для передачи управления по другому адресу. Если флаги SF и OF одинаковы (оба 0 или оба 1), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01111101|--disp--|

JL/JNGE: Переход по "меньше" или "не больше или равно"

О п е р а ц и я: Используется после проверки знаковых данных для передачи управления по другому адресу. Если флаги SF и OF различны, то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01111100|--disp--|

JLE/JNG: Переход по "меньше или равно" или "не больше"

О п е р а ц и я: Используется после проверки знаковых данных для передачи управления по другому адресу. Если флаг ZF равен единице (нулевое состояние) и флаги SF и OF различны, то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом

переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01111110|--disp--|

JMP: Безусловный переход

О п е р а ц и я: Выполняет переход по указанному адресу при любых условиях. Команда JMP заносит в регистр IP необходимый адрес перехода. Существует пять типов команды JMP для передачи управления внутри сегмента или между сегментами. При межсегментном переходе в регистр CS заносится также новый сегментный адрес.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д (пять форматов):

Прямой переход внутри сегмента:
|11101001|disp-low|disp-high|
Прямой переход внутри сегмента (короткий):
|11101011|--disp--|
Косвенный переход внутри сегмента:
|11111111|mod100r/m|
Косвенный межсегментный переход:
|11111111|mod101r/m|
Прямой межсегментный переход:
|11101010|offset-low|offset-high|seg-low|seg-high|

JNC: Переход если нет переноса

О п е р а ц и я: Идентична JAE/JNB.

JNE/JNZ: Переход по "не равно" или по "не нуль"

О п е р а ц и я: Используется после проверки знаковых данных для передачи управления по другому адресу. Если флаг ZF равен нулю (ненулевое состояние), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01110101|--disp--|

INO: Переход, если нет переполнения

О п е р а ц и я: Используется для передачи управления по определенному адресу после проверки на отсутствие переполнения. Если флаг OF равен, нулю (нет переполнения), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход. См. также команду JO.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01110001|--disp--|

JNP/JPO: Переход, если нет паритета или паритет нечетный

О п е р а ц и я: Приводит к передаче управления по определенному адресу, если в результате операции обнаружено отсутствие паритета или паритет нечетный. Нечетный паритет в данном случае означает, что в результате операции в . младших восьми битах получено нечетное число битов. Если флаг PF равен нулю (нечетный паритет), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход. См. также команду JP/JPE.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01111011|--disp--|

JHS: Переход, если нет знака

О п е р а ц и я: Приводит к передаче управления по определенному адресу, если в результате операции получен положительный знак. Если флаг SF равен нулю (положительное), то команда JNS прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход. См. также команду JS.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01111001|--disp--|

JO: Переход по переполнению

О п е р а ц и я: Приводит к передаче управления по определенному адресу, если в результате операции получено состояние переполнения. Если флаг OF равен единице (переполнение), то команда JO прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход. См. также команду JNO.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01110000|--disp--|

JP/JPE: Переход, если есть паритет или паритет четный

О п е р а ц и я: Приводит к передаче управления по определенному адресу, если в результате операции обнаружен четный паритет. Четный паритет в данном случае означает, что в результате операции в младших восьми битах получено четное число битов. Если флаг PF равен единице (четный паритет), то команда прибавляет к регистру IP значение операнда (относительное смещение) и выполняет таким образом переход. См. также команду JNP/JPO.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01111010|--disp--|

JS: Переход по знаку

О п е р а ц и я: Передает управления по определенному адресу, если в результате операции получен отрицательный знак. Если флаг SF равен единице (отрицательно), то команда JS прибавляет к регистру IP значение операнда

(относительное смещение) и выполняет таким образом переход. См. также команду JNS.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |01111000|--disp--|

LAHF: Загрузка флагов в регистр AH

О п е р а ц и я: Загружает значение флагового регистра в регистр AH. Данная команда обеспечивает совместимость с процессором 8080. См. также команду SAHF. Команда LAHF заносит правый байт флагового регистра в регистр AH в следующем виде:

SZ * A * P * C   (* обозначает неиспользуемые биты)

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 10011111 (без операндов)

LDS: Загрузка регистра сегмента данных

О п е р а ц и я: Инициализирует начальный адрес сегмента данных и адрес смещения к переменной для обеспечения доступа к данной переменной. Команда LDS загружает в необходимые регистры четыре байта из области памяти, содержащей относительный адрес и сегментный адрес. Сегментный адрес помещается в регистр DS, а относительный адрес - в любой из общих или индексных регистров или в регистровый указатель. Следующая команда загружает относительный адрес в регистр DI:

```
LDS   DI,адрес_памяти
```

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |11000101|modregr/m|

LES: Загрузка регистра дополнительного сегмента

О п е р а ц и я: Инициализирует начальный адрес дополнительного сегмента и адрес смещения к переменной для .обеспечения доступа к данной переменной. См. также команду LDS.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |11000100|modregr/m|

LOCK: Блокировка шины доступа к данным

О п е р а ц и я: Запрещает другим (сопроцессорам одновременно изменять элементы данных. Команда LOCK представляет собой однобайтовый префикс, который можно кодировать непосредственно перед любой командой. Данная операция посылает сигнал в другой процессор, запрещая использование данных, пока не будет завершена следующая команда.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 11110000

LODS/LODSB/LODSW: Загрузка однобайтовой или двухбайтовой строки

О п е р а ц и я: Загружает из памяти один байт в регистр AL или одно слово в регистр AX. Несмотря на то, что команда LODS выполняет строковую операцию, нет смысла использовать ее с префиксом REP. Регистровая пара DS:SI адресует в памяти байт (для LODSB) или слово (для LODSW), которые загружаются в регистр AL или AX соответственно. Если флаг DF равен нулю, то операция прибавляет 1 (для байта) или 2 (для слова) к регистру SI. Если флаг DF равен единице, то операция вычитает 1 (для байта) или 2 (для слова) из регистра SI.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 1010110w (без операндов).

LOOP: Цикл

О п е р а ц и я: Управляет выполнением группы команд определенное число раз. До начала цикла в регистр CX должно быть -загружено число выполняемых циклов. Команда LOOP находится в конце цикла, где она уменьшает значение в регистре CX. на единицу. Если значение в регистре CX не равно нулю, то команда передает управление по адресу, указанному в операнде (прибавляет к регистру IP значение операнда); в противном случае управление передается на следующую после LOOP команду (происходит выход из цикла).

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: у11100010у--disp--у

LOOPE/LOOPZ: Цикл, если равно или нуль

О п е р а ц и я: Управляет выполнением группы команд определенное число раз или пока установлен флаг ZF (в единичное состояние). Команды LOOPE/LOOPZ аналогичны команде LOOP, за исключением того, что по этим командам цикл прекращается либо по нулевому значению в регистре CX, либо по нулевому значению флага ZF (ненулевое состояние). См. также команды LOOPNE/LOOPNZ.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |11100001|--disp--|

LOOPNE/LOOPNZ: Цикл, если не равно или не нуль

О п е р а ц и я: Управляет выполнением группы команд определенное число раз или пока сброшен флаг ZF (в нулевое состояние)., Команды LOOPNE/LOOPNZ аналогичны команде LOOP за исключением того, что по этим командам цикл прекращается либо по нулевому значению в регистре CX, либо по единичному значению флага ZF (нулевое состояние). См. также команды LOOPE/LOOPZ.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: |11100000|--disp--|

MOV: Пересылка данных

Операция: Пересылает один байт или одно слово между регистрами или между регистром и памятью, а также передает непосредственное значение в регистр или в память. Команда MOV не может передавать данные между двумя адресами памяти (для этой цели служит команда MOVS). Существует семь типов команды MOV.

Флаги: Не меняются.

Объектный код (семь форматов):

```
Регистр/память в/из регистр:
    |100010dw|modregr/m|
Непосредственное значение в регистр/память:
    |1100011w|mod000r/m|--data--|data если w=1|
Непосредственное значение в регистр:
    |1011wreg|--data--|data если w=1|
Память в регистр AX (AL):
    |1010000w|addr-low|addr-high|
Регистр AX (AL) в память:
    |1010001w|addr-low|addr-high|
Регистр/память в сегментный регистр:
    |10001110|mod0sgr/m| (sg - сегментный регистр)
Сегментный регистр в регистр/память:
    |10001100|mod0sgr/m| (sg - сегментный регистр)
```

MOVS/MOVSB/MOVSW: Пересылка строки байт или строки слов

Операция: Пересылает данные между областями памяти. Команды MOVS(B/W) обычно используются с префиксом REP. Команда MOVSB пересылает любое число байтов, а команда MOVSW - любое число слов. Перед выполнением команды регистровая пара DS:SI должна адресовать источник пересылки ("откуда") а регистровая пара ES:DI - получатель пересылки ("куда"). Если флаг DF равен нулю, то операция пересылает данные слева направо и увеличивает регистры SI и DI. Если флаг DF равен единице то операция пересылает данные справа налево и уменьшает регистры SI и DI.

Флаги: Не меняются.

Объектный код: 1010010w (без операндов).

MUL: Беззнаковое умножение

Операция: Умножает беззнаковое множимое (8 или 16 бит) на беззнаковый множитель (8 или 16 бит). Левый единичный бит рассматривается как бит данных, но не как знак минус для отрицательных чисел. Для 8-битового умножения множимое должно находиться в регистре AL, а множитель возможен в регистре или в памяти, например MUL CL. Произведение получается в регистре AX. Для 16-битового умножения множимое должно находиться в регистре AX, а множитель возможен в регистре или в памяти, например MUL BX. Произведение получается в регистровой паре DX:AX. См. также команду IMUL.

Флаги: Команда воздействует на флаги CF и OF (флаги AF, PF, SF и ZF не определены).

Объектный код: |1111011w|mod100r/m|

NEG: Изменение знака числа

Операция: Меняет двоичное значение из положительного в отрицательное и из отрицательного в положительное. Команда NEG вычисляет двоичное дополнение от указанного операнда: вычитает операнд из нуля и прибавляет единицу. Операндом может быть байт или слово в регистре или в памяти. См. также команду NOT. Флаги: Команда воздействует на флаги AF, CF, OF, PF, SF и ZF.

Объектный код: |1111011w|mod011r/m|

NOP: Нет операции

Операция: Применяется для удаления или вставки машинных кодов или для задержки выполнения программы. Команда NOP выполняет операцию XCHG AX,AX, которая ничего не меняет.

Флаги: Не меняются.

Объектный код: 10010000 (без операндов)

NOT: Логическое НЕТ

Операция: Меняет нулевые биты на единичные и наоборот. Операндом может быть байт или слово в регистре или в памяти. См. также команду NEG.

Флаги: Не меняются.

Объектный код: |1111011w|mod010r/m|

OR: Логическое ИЛИ

Операция: Выполняет поразрядную дизъюнкцию (ИЛИ) над битами двух операндов. Операндами являются байты или слова в регистрах или в памяти, второй операнд может иметь непосредственное значение. Команда OR обрабатывает операнды побитово. Если любой из проверяемых бит равен единице, то бит в операнде 1 становится равным единице, в противном случае бит в операнде 1 не изменяется. См. также команды AND и XOR.

Флаги: Команда воздействует на флаги CF, OF, PF, SF и ZF (флаг AF неопределен).

Объектный код (три формата):

```
Регистр/память с регистром:
    |000010dw|modregr/m|
Непосредственное с аккумулятором:
    |0000110w|--data--|data если w=1|
Непосредственное с регистром/памятью:
    |1000000w|mod001r/m|--data--|data если w=1|
```

OUT: Вывод байта или слова в порт

Операция: Передает в выводной порт байт из регистра AL или

слово из регистра AX. Порт кодируется как фиксированный числовой операнд (OUT порт#,AX) или как переменная в регистре DX (OUT DX.AX). Процессоры 80186, 80286 и 80386 имеют, кроме того, команду OUTS (Output String - вывод строки). См. также команду IN.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д (два формата):

```
Переменный порт:   |1110111w|
Фиксированный порт:|1110011w|--port--|
```

POP: Извлечение слова из стека

О п е р а ц и я: Передает слово (помещенное ранее в стек) в указанный операнд. Регистр SP указывает на текущее слово в вершине стека. Команда POP извлекает слово из стека и увеличивает значение в регистре SP на 2. Существует три типа команды POP в зависимости от операнда: общий регистр, сегментный регистр, слово в памяти. См. также команду PUSH.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д (три формата):

```
Регистр:           |01011reg|
Сегментный регистр: |000sg111| (sg-сегм.рег.)
Регистр/память:     |10001111|mod000r/m|
```

POPA: Извлечение из стека всех общих регистров (80188,80186,80286)

О п е р а ц и я: Извлекает из стека восемь значений в регистры DI, SI BP, SP, BX, DX, CX, AX в указанной последовательности и увеличивает регистр SP на 16. Регистры обычно записываются в стек ранее соответствующей командой PUSHA.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 01100001 (без операндов).

POPF: Извлечение флагов из стека

О п е р а ц и я: Передает биты (помещенные ранее в стек) во флаговый регистр. См. также команды PUSHF и POP. Регистр SP указывает на текущее слово в вершине стека. Команда POPF передает биты из этого слова во флаговый регистр и увеличивает значение в регистре SP на 2. Обычно команда PUSHF записывает значения флагов в стек, а команда POPF восстанавливает эти флаги.

Ф л а г и: Команда воздействует на все флаги.

О б ъ е к т н ы й к о д: 10011101 (без операндов).

PUSH: Занесение слова в стек

О п е р а ц и я: Сохраняет значение слова (адрес или элемент данных) в стеке для последующего использования. Регистр SP указывает на текущее слово в вершине стека. Команда PUSH уменьшает значение в регистре SP на 2

и передает слово из указанного операнда в новую вершину стека. Существует три типа команды PUSH в зависимости от операнда: общий регистр, сегментный регистр или слово в памяти. См. также команды POP и PUSHF.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д (три формата):

```
Регистр:            |01010reg|
Сегментный  регистр: |000sg111| (sg-сегм.рег.)
Регистр/память:      |11111111|mod110r/m|
```

PUSHA: Занесение в стек всех общих регистров (80188,80186,80286)

О п е р а ц и я: Записывает в стек восемь значений регистров AX, CX, DX, BX, SP, BP, SI, DI в указанной последовательности и уменьшает регистр SP на 16. Обычно команда POPA позже восстановит эти регистры из стека.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 01100000 (без операндов).

PUSHF: Занесение флагов в стек

О п е р а ц и я: Сохраняет значения флагов из флагового регистра в стеке для последующего использования. Регистр SP указывает на текущее слово в вершине стека. Команда PUSHF уменьшает значение в регистре SP на 2 и передает флаги в новую вершину стека. См. также команды POPF и PUSH.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 10011100 (без операндов).

RCL и RCR: Циклический сдвиг влево через перенос и циклический сдвиг вправо через перенос

О п е р а ц и я: Выполняет циклический сдвиг битов (ротацию) влево или вправо через флаг CF. Данные операции могут выполняться в байте или в слове, в регистре или в памяти. Ротация на один бит кодируется в команде значением 1; ротация более чем на один бит требует указания регистра CL, который содержит счетчик. Для команды RCL значение флага CF записывается в бит 0, а. самый левый бит записывается во флаг CF; все другие биты сдвигаются влево. Для команды RCR значение флага CF записывается в самый левый бит, а бит 0 записывается во флаг CF; все другие биты сдвигаются вправо. См. также команды ROL и ROR.

Ф л а г и: Команда воздействует на флаги CF и OF.

```
О б ъ е к т н ы й  к о д:  RCL: |110100cw|mod010r/m|
                          RCR: |110100cw|mod011r/m|
                               (если c = 0, то сдвиг на 1;
                               если c = 1, то сдвиг на CL).
```

REP/REPE/REPZ/REPNE/REPNZ: Повтор строковой операции

О п е р а ц и я: Повторяет строковую операцию определенное число раз. Используется в качестве префикса повторения перед строковыми командами

CMPS, MOVS, SCAS, STOS. Счетчик повторений должен быть загружен в регистр CX до выполнения строковой команды. Операция уменьшает регистр CX на 1 при каждом выполнении строковой команды. Для префикса REP операция повторяется, пока содержимое регистра CX не достигнет нуля. Для префикса REPE/REPZ операция повторяется, пока регистр CX содержит ненулевое значение и флаг ZF равен 1 (нулевое состояние). Для префикса REPNE/REPNZ операция повторяется, пока регистр CX содержит ненулевое значение и флаг ZF равен 0 (ненулевое состояние).

Ф л а г и: Определяются соответствующей строковой командой.

О б ъ е к т н ы й к о д: REP/REPNE: 11110010
REPE: 11110011

RET: Возврат из процедуры

О п е р а ц и я: Возвращает управление из процедуры, вызванной ранее командой CALL. Команда CALL может передавать управление внутри одного сегмента или между сегментами. Команда RET заносит слово из вершины стека в регистр IP и увеличивает значение SP на 2. Для межсегментного возврата команда RET, кроме того, заносит слово из новой вершины стека в регистр CS и еще раз увеличивает значение SP на 2. Любой числовой операнд команды (например, RET 4) прибавляется к указателю стека SP.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д (четыре формата):

Внутри сегмента: |11000011|
Внутри сегмента с операндом: |11000010|data-low|data-high|
Между сегментами: |11001011|
Между сегментами с операндом: |11001010|data-low|data-high|

ROL и ROR: Циклический сдвиг влево и циклический сдвиг вправо

О п е р а ц и я: Выполняет циклический сдвиг битов (ротацию) влево или вправо. Данные операции могут выполняться в байте или в слове, в регистре или в памяти. Ротация на один бит кодируется в команде значением 1; ротация более чем на один бит требует указания регистра CL, который содержит счётчик. Для команды ROL самый левый бит записывается в бит 0; все другие биты сдвигаются влево. Для команды ROR бит 0 записывается в самый левый бит; все другие биты сдвигаются вправо. См. также команды RCL и RCR.

Ф л а г и: Команда воздействует на флаги CF и OF.

О б ъ е к т н ы й к о д: ROL: |110100cw|mod000r/m|
ROR: |110100cw|mod001r/m|
(если c = 0, то сдвиг на 1;
если c = 1, то сдвиг на CL).

SAHF: Установка флагов из регистра AH

О п е р а ц и я: Данная команда обеспечивает совместимость с процессором 8080 для пересылки значений флагов из регистра AH во флаговый регистр. См. также команду LAHF. Команда SAHF пересылает определенные биты из регистра AH во флаговый регистр в следующем виде:

SZ * A * P * C   (* обозначает неиспользуемые биты)

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 10011110 (без операндов)

SAL, SAR, SHL и SHR: Сдвиг влево или вправо

О п е р а ц и я: Выполняет сдвиг битов влево или вправо. Данные операции могут выполняться в байте или в слове, в регистре или в памяти. Сдвиг на один бит кодируется в команде значением 1; сдвиг более чем на один бит требует указания регистра CL, который содержит счетчик сдвига. Команда SAR выполняет арифметический сдвиг, который учитывает знак сдвигаемого значения. Команды SHL и SHR выполняют логический сдвиг и рассматривают знаковый бит как обычный бит данных. Команда SAL выполняется аналогично команде SHL. Команды SAL и SHL сдвигают биты влево определенное число раз и правый освобождающийся бит заполняют нулевым значением. Команда SHR сдвигает биты вправо определенное число раз и левый освобождающийся бит заполняет нулевым значением. Команда SAR сдвигает биты вправо определенное число раз и левый освобождающийся бит заполняет значением знакового бита (0 или 1). Во всех случаях значения битов, выдвигаемых за разрядную сетку, теряются.

Ф л а г и: Команда воздействует на флаги CF, OF, PF, SF и ZF (флаг AF не определен).

```
О б ъ е к т н ы й  к о д:  SAL/SHL: |110100cw|mod100r/m|
                           SAR:     |110100cw|mod111r/m|
                           SHR:     |110100cw|mod101r/m|
                                     (если c = 0, то сдвиг на 1;
                                      если c=1, то сдвиг на CL);
```

SBB: Вычитание с заемом

О п е р а ц и я: Обычно используется при вычитании многословных двоичных величин для учета единичного бита переполнения в последующей фазе операции. Если флаг CF установлен в 1, то команда SBB сначала вычитает 1 из операнда 1. Команда SBB всегда вычитает операнд 2 из операнда 1, аналогично команде SUB. См. также команду ADC.

Ф л а г и: Команда воздействует на флаги AF, CF, OF, PF. SF и ZF.

О б ъ е к т н ы й к о д (три формата):

```
Регистр из регистра или памяти: |000110dw|modregr/m|
Непосредственное значение из регистра AX (AL):
    |0001110w|--data--|data, если w=1|
Непосредственное значение из регистра или памяти:
    |100000sw|mod011r/m|--data--|data, если sw=01|
```

SCAS/SCASB/SCASW: Поиск байта или слова в строке

О п е р а ц и я: Выполняет поиск определенного байта или слова в строке. Для команды SCASB необходимое значение загружается в регистр AL, а для команды SCASW - в регистр AX. Регистровая пара ES:DI указывает на строку в памяти, которая должна быть сканирована. Данные команды обычно используются с префиксом REPE или REPNE. Если флаг DF равен нулю, то операция сканирует память слева направо и увеличивает регистр DI. Если

флаг DF равен единице, то операция сканирует память справа налево и уменьшает регистр DI.

Ф л а г и: Команда воздействует на флаги AF, CF, OF, PF, SF и ZF.

О б ъ е к т н ы й к о д: 1010111w (без операндов).

STC: Установка флага переноса

О п е р а ц и я: Устанавливает значение флага CF в 1. См. также команду STD.

Ф л а г и: Команда воздействует на флаг CF (устанавливается в 1).

О б ъ е к т н ы й к о д: 11111001 (без операндов).

STD: Установка флага направления

О п е р а ц и я: Устанавливает значение флага направления в 1. В результате строковые операции, такие, как MOVS или CMPS, обрабатывают данные справа налево. См. также CLD.

Ф л а г и: Команда воздействует на флаг DF (устанавливается в 1).

О б ъ е к т н ы й к о д: 11111101 (без операндов).

STI: Установка флага прерывания

О п е р а ц и я: Разрешает маскируемые внешние прерывания после выполнения следующей команды и устанавливает значения флага прерывания IF в 1. См. также команду CLI.

Ф л а г и: Команда воздействует на флаг IF (устанавливается в 1).

О б ъ е к т н ы й к о д: 11111011 (без операндов).

STOS/STOSB/STOSW: Запись однобайтовой или двухбайтовой строки

О п е р а ц и я: Сохраняет байт или слово в памяти. При использовании префикса REP операция дублирует значение байта или слова определенное число раз, что делает ее удобной для очистки областей памяти. Для команды STOSB необходимый байт загружается в регистр AL, а для команды STOSW необходимое слово загружается в регистр AX. Регистровая пара ES:DI указывает область памяти, куда должен быть записан байт или слово. Если флаг DF равен нулю, то операция записывает в память слева направо и увеличивает регистр DI. Если флаг DF равен единице, то операция записывает в память справа налево и уменьшает значение в регистре DI.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 1010101w (без операндов).

SUB: Вычитание двоичных чисел

О п е р а ц и я: Вычитает байт или слово в регистре, памяти или непосредственное значение из регистра; или вычитает байт или слово в

регистре или непосредственное значение из памяти. См. также команду SBB.

Ф л а г и: Команда воздействует на флаги AF, CF, OF, PF, SF и ZF.

О б ъ е к т н ы й к о д (три формата):

```
    Регистр из регистра или памяти:
        |001010dw|modregr/m|
    Непосредственное значение из регистра AX (AL):
        |0010110w|--data--|data, если w=1|
    Непосредственное значение из регистра или памяти:
        |100000sw|mod101r/m|--data--|data, если sw=01|
```

TEST: Проверка битов

О п е р а ц и я: Команда выполняет проверку байта или слова на определенную битовую комбинацию. Команда TEST действует аналогично команде AND, но не изменяет результирующий операнд. Операнды могут иметь однобайтовые или двухбайтовые значения. Второй операнд может иметь непосредственное значение. Команда выставляет флаги в соответствии с логической функцией И.

Ф л а г и: Команда воздействует на флаги CF, OF, PF, SF и ZF (флаг AF не определен).

О б ъ е к т н ы й к о д (три формата):

```
    Регистр или память И регистр:
        |1000010w|modregr/m|
    Непосредственное значение И регистр AX (AL):
        |1010100w|--data--|data, если w=1|
    Непосредственное значение И регистр или память:
        y1111011w|mod000r/m|--data--|data, если w=1|
```

WAIT: Установка процессора в состояние ожидания

О п е р а ц и я: Позволяет процессору оставаться в состоянии ожидания, пока не произойдет внешнее прерывание. Данная операция необходима для обеспечения синхронизации процессора с внешним устройством или с сопроцессором. Процессор ожидает, пока внешнее устройство (или сопроцессор) не закончит выполнение операции и на входной линии TEST не появится сигнал (активный уровень).

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 10011011

XCHG: Перестановка

О п е р а ц и я: Переставляет два байта или два слова между двумя регистрами (например, XCHG AH,BL) или между регистром и памятью (например, XCHG CX,word).

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д (два формата):

```
    Регистр и аккумулятор:    |10010reg|
```

Регистр/память и регистр: |1000011w|modregr/m|

XLAT: Перекодировка

О п е р а ц и я: Транслирует байты в другой формат, например при переводе нижнего регистра в верхний или при перекодировке ASCII-кода в EBCDIC-код. Для выполнения данной команды необходимо определить таблицу преобразования байт и загрузить ее адрес в регистр BX. Регистр AL должен содержать байт, который будет преобразован с помощью команды XLAT. Операция использует значение в регистре AL как смещение в таблице, выбирает байт по этому смещению и помещает его в регистр AL.

Ф л а г и: Не меняются.

О б ъ е к т н ы й к о д: 11010111 (без операндов).

XOR: Исключающее ИЛИ

О п е р а ц и я: Выполняет логическую операцию исключающего ИЛИ над битами двух операндов. Операндами являются байты или слова в регистрах или в памяти, второй операнд может иметь непосредственное значение. Команда XOR обрабатывает операнды побитово. Если проверяемые биты одинаковы, то команда XOR устанавливает бит в операнде 1 равным нулю, если биты различны, то бит в операнде 1 устанавливается равным единице. См. также команды AND и OR.

Ф л а г и: Команда воздействует на флаги CF, OF, PF, SF и ZF (флаг AF неопределен).

О б ъ е к т н ы й к о д (три формата):

Регистр/память с регистром:
|001100dw|modregr/m|
Непосредственное с регистром AX (AL):
|0011010w|--data--|data если w=1|
Непосредственное с регистром/памятью:
|1000000w|mod110r/m|--data--|data если w=1|

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1: Коды ASCII-символов

Ниже представлены первые 128 символов ASCII-кода. В руководстве по языку BASIC приведены остальные 128 символов. На помним, что шест.20 представляет стандартный символ пробела.

Таблица A-1. Набор ASCII символов

Дес	Шест	Симв	Дес	Шест	Симв	Дес	Шест	Симв	Дес	Шест	Симв
000	00h	Нуль	032	20h	sp	064	40h	@	096	60h	`
001	01h	Начало заголовка	033	21h	!	065	41h	A	097	61h	a

002	02h	Начало текста	034	22h	"	066	42h	B	098	62h	b
003	03h	Конец текста	035	23h	#	067	43h	C	099	63h	c
004	04h	Конец передачи	036	24h	$	068	44h	D	100	64h	d
005	05h	КТМ	037	25h	%	069	45h	E	101	65h	e
006	06h	Да	038	26h	&	070	46h	F	102	66h	f
007	07h	Звонок	039	27h	'	071	47h	G	103	67h	g
008	08h	Возврат на шаг	040	28h	(	072	48h	H	104	68h	h
009	09h	Гориз.табуляция	041	29h	)	073	49h	I	105	69h	i
010	0Ah	Перевод строки	042	2Ah	*	074	4Ah	J	106	6Ah	j
011	0Bh	Верт.табуляция	043	2Bh	+	075	4Bh	K	107	6Bh	k
012	0Ch	Перевод страницы	044	2Ch	,	076	4Ch	L	108	6Ch	l
013	0Dh	Возврат каретки	045	2Dh	-	077	4Dh	M	109	6Dh	m
014	0Eh	Shift out	046	2Eh	.	078	4Eh	N	110	6Eh	n
015	0Fh	Shift in	047	2Fh	/	079	4Fh	O	111	6Fh	o
016	10h	Data line esc	048	30h	0	080	50h	P	112	70h	p
017	11h	Управление 1	049	31h	1	081	51h	Q	113	71h	q
018	12h	Управление 2	050	32h	2	082	52h	R	114	72h	r
019	13h	Управление 3	051	33h	3	083	53h	S	115	73h	s
020	14h	Управление 4	052	34h	4	084	54h	T	116	74h	t
021	15h	Нет	053	35h	5	085	55h	U	117	75h	u
022	16h	Синхронизация	054	36h	6	086	56h	V	118	76h	v
023	17h	Конец блока	055	37h	7	087	57h	W	119	77h	w
024	18h	Анулирование	056	38h	8	088	58h	X	120	78h	x
025	19h	End of medium	057	39h	9	089	59h	Y	121	79h	y
026	1Ah	Замена	058	3Ah	:	090	5Ah	Z	122	7Ah	z
027	1Bh	Escape	059	3Bh	;	091	5Bh	[	123	7Bh	{
028	1Ch	Раздел.файла	060	3Ch	<	092	5Ch	\	124	7Ch	|
029	1Dh	Раздел.группы	061	3Dh	=	093	5Dh		125	7Dh	}
030	1Eh	Раздел.записи	062	3Eh	>	094	5Eh	^	126	7Eh	~
031	1Fh	Раздел.единицы	063	3Fh	?	095	5Fh	_	127	7Fh	Забой

ПРИЛОЖЕНИЕ 2: Шестнадцатерично-десятичные преобразования

В данном приложении представлены приемы преобразования между шестнадцатеричным и десятичным форматами. В первом разделе показан пример преобразования шест. A7B8 в десятичное 42936, а во втором - 42936 обратно в шест. A7B8.

Преобразование шестнадцатеричного формата в десятичный

Для перевода шест. A7B8 в десятичное число необходимо последовательно, начиная с самой левой шест. цифры (A), умножать на 16 и складывать со следующей цифрой. Так как операции выполняются в десятичном формате, то шест. числа от A до F необходимо преобразовать в десятичные от 10 до 15.

```
Первая цифра: A (10)               10
Умножить на 16                    x16
                                  ---
                                  160
Прибавить следующую цифру, 7       +7
                                  ---
                                  167
Умножить на 16                    x16
                                 ----
                                 2672
Прибавить следующую цифру, B (11)  +11
                                 ----
```

2683
Умножить на 16 x16

42928
Прибавить следующую цифру, 8 +8

Десятичное значение 42936

Можно использовать также таблицу преобразования. Для шест. числа A7B8 представим правую цифру (8) как позицию 1, следующую влево цифру (B) как позицию 2, следующую цифру (7) как позицию 3 и самую левую цифру (A) как позицию 4. Из таблицы B-1 выберем значения для каждой шест. цифры:

Для позиции 1 (8), столбец 1 8
Для позиции 1 (8), столбец 1 176
Для позиции 1 (8), столбец 1 1792
Для позиции 1 (8), столбец 1 40960

Десятичное значение 42936

Преобразование десятичного формата в шестнадцатеричный

Для преобразования десятичного числа 42936 в шестнадцатеричный формат необходимо сначала исходное число 42936 разделить на 16; число, получившееся в остатке, (6) является младшей шестнадцатеричной цифрой. Затем полученное частное необходимо снова разделить на 16 и полученный остаток (11 или B) дает следующую влево шестнадцатеричную цифру. Продолжая таким образом деления до тех пор, пока в частном не получится 0, получим из остатков все необходимые шестнадцатеричные цифры.

	Частное	Остаток	Шест.	
42936 / 16	2683	8	8	(младшая цифра)
2683 / 16	167	11	B	
167 / 16	10	7	7	
10 / 16	0	10	A	(старшая цифра)

Для преобразования чисел из десятичного формата в шестнадцатеричный можно также воспользоваться таблицей B-1. Для десятичного числа 42936 необходимо найти в таблице число равное или ближайшее меньшее исходному, и записать соответствующую шестнадцатеричную цифру и ее позицию. Затем следует вычесть найденное десятичное число из 42936 и с полученной разностью проделать проделать ту же операцию:

Дес. Шест.

Исходное десятичное число 42936
Вычесть ближайшее меньшее 40960 A000

Разность 1976
Вычесть ближайшее меньшее 1792 700

Разность 184
Вычесть ближайшее меньшее 176 B0

Разность 8 8

Результирующее шест. число A7B8

Таблица B-1. Шестнадцатерично-десятичные преобразования

ШЕСТ	ДЕС	ШЕСТ	ДЕС	ШЕСТ	ДЕС	ШЕСТ	ДЕС	ШЕСТ	ДЕС	ШЕСТ	ДЕС	ШЕСТ	ДЕС	ШЕСТ	ДЕС
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268435456	1	16777216	1	1048576	1	65536	1	4096	1	256	1	16	1	1
2	536870912	2	33554432	2	2097152	2	131072	2	8192	2	512	2	32	2	2
3	805306368	3	50331648	3	3145728	3	196608	3	12288	3	768	3	48	3	3
4	1073741824	4	67108864	4	4194304	4	262144	4	16384	4	1024	4	64	4	4
5	1342177280	5	83886080	5	5242880	5	327680	5	20480	5	1280	5	80	5	5
6	1610612736	6	100663296	6	6291456	6	393216	6	24576	6	1536	6	96	6	6
7	1879048192	7	117440512	7	7340032	7	458752	7	28672	7	1792	7	112	7	7
8	2147483648	8	134217728	8	8388608	8	524288	8	32768	8	2048	8	128	8	8
9	2415919104	9	150994944	9	9437184	9	589824	9	36864	9	2304	9	144	9	9
A	2684354560	A	167772160	A	10485760	A	655360	A	40960	A	2560	A	160	A	10
B	2952790016	B	184549376	B	11534336	B	720896	B	45056	B	2816	B	176	B	11
C	3221225472	C	201326592	C	12582912	C	786432	C	49152	C	3072	C	192	C	12
D	3489660928	D	218103808	D	13631488	D	851968	D	53248	D	3328	D	208	D	13
E	3758096384	E	234881024	E	14680064	E	917504	E	57344	E	3584	E	224	E	14
F	4026531840	F	251658240	F	15728640	F	983040	F	61440	F	3840	F	240	F	15
8		7		6		5		4		3		2		1	

Шестнадцатиричные позиции

ПРИЛОЖЕНИЕ 3: Зарезервированные слова

Большинство из следующих зарезервированных слов при использовании их для определении элементов данных могут привести к ошибкам ассемблирования (в ряде случаев - к весьма грубым):

Имена регистров

```
AH   BH   CH   DH   CS   SS   BP
AL   BL   CL   DL   DS   SI   SP
AX   BX   CX   DX   ES   DI
```

Мнемокоды

```
AAA    DIV    JLE    JS      OR     SBB
AAD    ESC    JMP    JZ      OUT    SCAS
AAM    HLT    JNA    LAHF    POP    SHL
AAS    IDIV   JNAE   LDS     POPF   SHR
ADC    IMUL   JNB    LEA     PUSH   STC
ADD    IN     JNBE   LES     PUSHF  STD
AND    INC    JNE    LOCK    RCL    STI
CALL   INT    JNG    LODS    RCR    STOS
CBW    INTO   JNGE   LOOP    REP    SUB
CLC    IRET   JNL    LOOPE   REPE   TEST
CLD    JA     JNLE   LOOPNE  REPNE  WAIT
CLI    JAE    JNO    LOOPNZ  REPNZ  XCHG
CMC    JB     JNP    LOOPZ   REPZ   XLAT
CMP    JBE    JNS    MOV     RET    XOR
CMPS   JCXZ   JNZ    MOVS    ROL
```

```
CWD      JE       JO       MUL      ROR
DAA      JG       JP       NEG      SAHF
DAS      JGE      JPE      NOP      SAL
DEC      JL       JPO      NOT      SAR
```

Директивы ассемблера

```
ASSUME   END      EXTRN    IFNB     LOCAL    PURGE
COMMENT  ENDIF    GROUP    IFNDEF   MACRO    RECORD
DB       ENDM     IF       IF1      NAME     REPT
DD       ENDP     IFB      IF2      ORG      SEGMENT
DQ       ENDS     IFDEF    INCLUDE  OUT      STRUC
DT       EQU      IFDIF    IRP      PAGE     SUBTTL
DW       EVEN     IFE      IRPC     PROC     TITLE
ELSE     EXITM    IFIDN    LABEL    PUBLIC
```

Прочие элементы языка

```
BYTE     FAR      LENGTH   MOD      PRT      THIS
COMMENT  GE       LINE     NE       SEG      TYPE
CON      GT       LT       NEAR     SHORT    WIDTH
DUP      HIGH     LOW      NOTHING  SIZE     WORD
EQ       LE       MASK     OFFSET   STACK
```

ПРИЛОЖЕНИЕ 4: РЕЖИМЫ АССЕМБЛИРОВАНИЯ И КОМПОНОВКИ

Ассемблирование программы

На дискете с ассемблерным пакетом имеются две версии ассемблера: ASM.EXE - сокращенная версия, в которой отсутствуют некоторые возможности, и MASM.EXE - полная версия, имеющая все возможности и являющаяся предпочтительной. Для ассемблирования вставьте ассемблерную дискету в дисковод A, а дискету с исходной программой - в дисковод B. Кто имеет винчестер или RAM-диск (виртуальный диск в памяти), могут в следующих примерах заменить соответствующие номера дисководов. Введите команды MASM (или ASM), что приведет к загрузке программы ассемблера с диска в память. На экране появится:

```
source filename [.ASM]:
object filename [filename.OBJ]:
source listing  [NUL.LST]:
cross-reference [NUL.CRF]:
```

Курсор при этом расположится в конце первой строки, где необходимо указать имя файла. Введите номер дисковода (если он не определен по умолчанию) и имя файла. Не следует набирать тип файла ASM - ассемблер подразумевает это. Во втором запросе предполагается аналогичное имя файла (но можно его заменить). Если необходимо, введите номер дисковода. Третий запрос предполагает, что листинг ассемблирования программы не требуется. Последний запрос предполагает, что листинг перекрестных ссылок не требуется. Для получения листинга на дисководе B введите B:.

Если вы хотите оставить значения по умолчанию, то в трех последних запросах просто нажмите Return. Ниже приведен пример запросов и ответов, в результате которых ассемблер должен создать OBJ-, LST- и CRF-файлы на дисководе B:

```
source filename [.ASM]:B:ASMPROG [Return]
object filename [filename.OBJ]:B: [Return]
source listing  [NUL.LST]:B: [Return]
cross-reference [NUL.CRF]:B: [Return]
```

Альтернативный вариант вызова ассемблера:

```
MASM B:name.ASM,B:name.OBJ,B:name.LST,B:name.CRF
```

В следующей команде используются значения по умолчанию:

```
MASM B:filename,B:,B:,B:;
```

- первый параметр B:filename идентифицирует исходную программу, которая должна быть ассемблирована;
- второй параметр B: указывает ассемблеру на необходимость записать объектный модуль на дисковод B под именем filename.OBJ;
- третий параметр B: указывает ассемблеру на необходимость записать листинг ассемблирования на дисковод B под именем filename.LST;
- четвертый параметр B: приводит к генерации файла перекрестных ссылок под именем filename.CRF;
- Точка с запятой в конце команды указывает ассемблеру на необходимость игнорировать любые последующие параметры. Хотя в приведенном примере таких параметров нет, кодирование точки с запятой является хорошим правилом для других вариантов команды.

При наличии ошибок ассемблирования программа ASM выдает только коды ошибок, которые объяснены в руководстве по Ассемблеру, в то время как программа MASM выдает и коды ошибок, и пояснения к ним. Всего имеется около 100 сообщений об ошибках.

Режимы ассемблирования включают следующие:

MASM /A   организовать исходные сегменты в алфавитном порядке;
MASM /D   генерировать LST-файл в обоих проходах ассемблирования;
MASM /E   ассемблировать команды для сопроцессора 8087 и генерировать связи для языка ФОРТРАН и пакета эмуляции языка ПАСКАЛЬ;
MASM /N   отменить генерацию таблицы идентификаторов;
MASM /R   ассемблировать команды для сопроцессора 8087 (аналогично /E);
MASM /S   сохранить исходные сегменты в оригинальной последовательности.

Файл перекрестных ссылок

Генерируемый в ходе ассемблирования CRF-файл может использоваться для создания листинга перекрестных ссылок на метки, идентификаторы и переменные в программе. Для получения CRF-фала необходимо на четвертый запрос ассемблера ответить, например. B:, полагая, что файл должен быть создан на диске B:

```
cross-reference [NUL.CRF]:B: [Return]
```

Для преобразования полученного CRF-файла в удобно отсортированную таблицу перекрестных ссылок используется программа CREF, имеющаяся на ассемблерном диске. Введите команду CREF; на экране появится два запроса:

```
Cref filename [.CRF]:
List filename [cross-ref.REF]:
```

На первый запрос введите имя CRF-файла, например B:ASMPROG. На второй запрос можно ввести только номер дисковода и получить имя по умолчанию.

Такой выбор приведет к преобразованию CRF-файла в файл перекрестных ссылок по имени filename.REF на дисководе B. Можно также кодировать всю команду в одной строке, например:

CREF B: filename. CRF, CON или CREF B:filename.CRF,B:

В первом примере файл перекрестных ссылок выводится на экран. Второй пример записывает файл по имени filename.REF на диск B.

Компоновка программ

Для компоновки EXE-программы вставьте дискету DOS в дисковод A, а дискету с программой - в дисковод B. Те, кто имеют винчестер или RAM-диск (виртуальный диск в памяти), могут в следующих примерах   заменить соответствующие номера дисководов. Простейший способ запуска компоновщика представляет ввод команды LINK, что приводит к загрузке компоновщика с диска в память. На экране появятся следующие запросы:

```
Object Modules [.OBJ]:
Run File [d:filename.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
```

Курсор при этом расположится в конце первой строки, где необходимо указать имя файла. Введите номер дисковода (если он не определен по умолчанию) и имя файла. Не следует набирать тип файла OBJ - компоновщик подразумевает это. Во втором запросе предполагается аналогичное имя файла (но можно его заменить). Если необходимо, введите номер дисковода. Третий запрос предполагает, что листинг компоновки программы не требуется. Последний запрос имеет отношение к библиотечным средствам DOS.

Если вы хотите оставить значения по умолчанию, то в трех последних запросах просто нажмите Return. Ниже приведен пример запросов и ответов, в результате которых компоновщик должен создать файлы EXE и CON:

```
Object Modules  [.OBJ]:B:ASMPROG [return]
Run File [d:filename.EXE]:B:  [return]
List File [NUL.MAP]:CON  [return]
Libraries [.LIB]:  [return]
```

Ответ CON приводит к выводу карты компоновки на экран для проверки на наличие сообщений об ошибках. Команду LINK можно сформировать также следующим образом:

```
LINK B:ASMPROG,B:,CON
```

Описание остальных режимов компоновщика LINK можно   найти   в руководстве no DOS.

Программа EXE2BIN

Программа EXE2BIN.COM в операционной системе DOS используется для преобразования EXE-модулей в COM-модули, при этом полагается,   что программы подготовлены в соответствии с требованиями для COM-модулей. Для запуска программы EXE2BIN вставьте дискету DOS в дисковод A, а дискету с EXE-файлом - в дисковод B. Для винчестерского диска следует использовать дисковод C. Введите следующую команду:

```
EXE2BIN B:filename,B:filename.COM
```

Первым операндом является имя EXE-файла, которое вводится без указания типа. Второй операнд представляет собой имя COM-файла; здесь можно изменить имя программы, но обязательно следует указывать тип COM. После того как преобразование будет выполнено, можно удалить файлы OBJ и EXE.

Ответы на некоторые вопросы

1.1. а) 01001101; в) 00111111.

1.2. а) 0101; в) 10101.

1.3. а) 00100010; в) 00100000.

1.4. а) 11101101; в) 11000110.

1.5. а) 00111000; в) 10000000.

1.6. а) 51; в) 5D.

1.7. а) 23C8; в) 8000.

1.8. а) 13; в) 59; д) FFF.

1.9. а) Стек, данные и код; в) параграф.

1.l0. а) AX, BX, CX, DX, Dl, SI; в) AX и DX; д) флаги.

1.11.а) ROM (постоянная память выполняет процедуры начальной загрузки и управляет операциями ввода-вывода.
б) RAM (оперативная память) представляет собой область, где находятся программы и данные при выполнении.

2.1. а) B82946.

2.2. E CS:101 54.

2.3. а)
```
   MOV       AX, 3004
   ADD       AX,3000
   RET
```
в) R и IP устанавливают IP в 0.

2.4. Произведение равно 0612H.

2.6. Использовать команду N для указания имени программы (например, COM), установить длину программы в регистре CX и с помощью команды W записать программу.

3.1. TITLE и PAGE.

3.2. а), б) и в) корректны; г) неверно, так как начинается с цифры;
д) корректно только при ссылке на регистр AX.

3.4. а) END; в) ENDS.

3.6. RET является командой, которая обеспечивает возврат в операционную систему; END является директивой, которая указывает ассемблеру, что в тексте более нет команд для ассемблирования.

3.7. ASSUME CS:CDSEG,DS:DATSEG,SS:STKSEG.

```
3.8. PUSH     DS
     SUB      AX, AX
     PUSH     AX
```

4.1. а) Ввести команду MASM и ответить на запросы: B:TEMPY B:, B: и B:

4.2. а) DEBUG B:TEMPY.EXE; б) B:TEMPY

4.3. а) Резервная копия исходного файла;
в) файл листинга ассемблирования с исходным и объектным кодом;
д) файл объектного кода;
ж) карта компоновки.

```
4.4. MOV      AX,DATSEG
     MOV      DS,AX
```

```
4.5. MOV      AL,30H
     SHL      AL,1
     MOV      BL,18H
     MUL      BL
```

4.6. Сегмент данных должен выглядеть следующим образом:

```
     FLDA     OB     28H
     FLDB     DB     14H
     FLDC     DW     ?
```

5.1. а) 2; в) 10; д) 8.

```
5.2.    CONAME    DB      'RGB Electronics'
```

```
5.3. а)  FLDA     DD      73H
     в)  FLDC     DW      ?
     д)  FLDE     DW      16, 19, 20, 27, 30
```

5.4. а) ASCII 3236; б) 1AH.

5.5. а) 26; в) 3A732500.

```
5.6. а)  MOV      AX,320
     в)  ADD      BX,40H
     д)  SHL      FLDB,1 (или SAL)
```

6.1. 64 К.

6.3. Это использует старшую область COM-программы или при недостатке памяти - старшие адреса оперативной памяти.

6.4. EXE2BIN SAMPLE,SAMPLE.COM.

7.1. +127 и -128. Операнд представляет собой однобайтовое значение от 00H до 7FH (от 0 до + 127) и от 80H до 0FFH (от -128 до -1).

7.2. а) 62B; в) 5EA (преобразует C6 в двоичное дополнение).

7.3. Приводим одно из многих возможных решений:

```
          MOV   AX,00
          MOV   BX,01
```

```
        MOV   CX,12
        MOV   DX,00
    B20:
        ADD   AX,BX        ;Номер в AX
        MOV   BX,DX
        MOV   DX,AX
        LOOP  B20
```

7.4. а) CMP DX,CX   в) JCXZ адрес   д) CMP BX,AX
JA адрес   или CMP CX,0   JLE или JNG
JZ адрес

7.5. а) OF (1); в) ZF (1); д) DF (1)

7.7. Первая (основная) процедура PROC должна иметь атрибут FAR, так как ее вызов и выполнение происходят из операционной системы. Атрибут NEAR указывает на то, что адрес процедуры находится внутри конкретной программы (т.е. в пределах одного сегмента ассемблерного модуля).

7.9. Три (по одному на каждый вызов CALL).

7.10.а) 1001 1010; в) 1111 1011; д) 0001 1100.

7.12.а) 01011100 11011100; в) 11001101 11001000.

8.1. Строка = 18H, столбец = 27H.

8.3.

```
      MOV     AX,0600H         ;Запрос
      MOV     BH,07            ; функции
      MOV     CX,OCOOH         ; очистки
      MOV     DX,164FH         ; экрана
      INT     10H
```

8.4. а) Вызов базовой функции DOS:

```
  MSSGE   DB      'What Is the date (mm/dd/yy)?',07H,'$'
        MOV     AH,09          ;Запрос функции
        MOV     DX,MSSGE       ;вывода сообщения
        INT     21H
```

8.5. а) Вызов базовой функции DOS:

```
      DATEPAR   LABEL  BYTE
      MAXLEN    DB  9          ;Длина области ввода
      ACTLEN    DB  ?
      DATEFLD   DB  9 DUP(' ')
           DB  '$'
           MOV AH,0AH          ;Запрос функции
           LEA DX,DATEPAR      ; ввода
           INT 21H
```

9.1. а) 1000 0001; в) 0111 1000.

9.2. а)

```
      MOV    AH,00        ;Запрос функции установки режима
      MOV    AL,02        ; 80x25 BW
      INT    10H
```

в)

```
      MOV    AH,060AH     ;Запрос прокрутки на 10 строк
      MOV    BH,07        ;Нормальный видеоатрибут
      MOV    CX,0000      ;Полный экран
      MOV    DX,184FH
      INT    10H
```

9.3. а) 48; в) 47.

9.5. Использовать команду INT 10H для ввода и проверить скэнкод.

10.1. Восемь цветов для фона и 16 цветов для текста.

10.2. а) 1011 0101; в) 1000 1100.

10.3. Низкое разрешение: четыре бита на пиксел дает 16 цветов.
Среднее разрешение: два бита на пиксел дает 4 цвета.
Высокое разрешение: один бит на пиксел дает 2 "цвета" (BW - черно-белое изображение).

10.4.
```
         MOV  AH,09          ;Функция вывода
         MOV  AL,04          ;Символ
         MOV  BH,00          ;Страница #0
         MOV  BL,01011010B   ;Светло-зеленый на сиреневом
         MOV  CX,05          ;Повторение 5 раз
         INT  10H
```

10.6. Сначала установить графический режим, а затем, используя

```
         INT  10H (AH=0BH),
```

установить цвет фона.

10.7. Сначала установить графический режим.

```
         MOV  AH,0DH         ;Прочитать точку
         MOV  CX,13          ;Столбец
         MOV  DX,12          ;Строка
         INT  10H
```

11.1. а)
```
              JCXZ   label2
      label1: MOV    AX,[SI]
              MOV    [DI],AX
              INC    DI
              INC    DI
              INC    SI
              INC    SI
              LOOP   label1
      label2: ...
```

11.2. Установить флаг направления DF для пересылки справа налево. Для команды MOVSB начальная установка на адреса NAME1 + 9 и NAME2 + 9. Для команды MOVSW начальная установка на адреса NAME1 + 8 и NAME2 + 8. Подпрограмма по метке H10SCAS может использовать

```
         MOV  AX,'mb'
```

и с помощью команды SCASW выполнить сканирование.

11.3. а)
```
         CLD                  ;Пересылка слева направо
         LEA    SI,CONAME     ;Инициализация
         LEA    DI,PRLINE     ; для пересылки
         MOV    CX,20         ; 20 байт
         REP    MOVSB
```

в)
```
         CLD                  ;Пересылка слева направо
         LEA    SI,CONAME+2   ;Начать с 3-го
```

```
        LODSW               ;Загрузить два байта

    д)      CLD                 ;Пересылка слева направо
        MOV    CX,20         ;20 байт
        LEA    SI,CONAME     ;Инициализация
        LEA    DI,PRLINE     ;адресов
        REPE   CMPSB         ;Сравнение
```

11.4. Одно из решений:

```
    H10SCAS  PROC   NEAR
        CLD                 ;Пересылка слева направо
        MOV    CX,10         ;10 байт
        LEA    DI,NAME1      ;Инициализация адреса
        MOV    AL,'e'       ; и символа для поиска
    H20:
        REPNE SCASB          ;Сканирование
        JNE    H3Q           ;Символ найден?
        CMP    BYTE PTR[DI],'r'  ; Да - следующий
        JNE    H20           ; байт равен r?
        MOV    AH, 03
    H30:    RET
    H10SCAS  ENDP
```

```
11.5. PATTERN  DB     03H,04H,05H,0B4H
    DISPLAY  DB     80 DUP(' '),'$'
        CLD                 ;Пересылка слева направо
        LEA    SI,PATTERN    ;Инициализация
        LEA    DI,DISPLAY    ; адресов
        MOV    CX,20         ;20 байт
        REP MOVSW            ;Переслать образец
```

Затем можно использовать команду INT 21 H для вывода на экран значения из области DISPLAY.

```
12.1.а)      MOV    AX,DATAY
        ADD   AX,DATAX
        MOV    DATAY,AX
```

б) См. рис.12.2 (многословное сложение).

12.2.Команда STC устанавливает флаг CF.Сумма равна 0148H плюс 0237H плюс 1

```
12.3.а)      MOV  AX,DATAX
        MUL  DATAY
```

б) См. рис.12.4 для умножения двойного слова на слово.

```
12.5. а)      MOV  AX,DATAX
        MOV  BL,25
        DIV  BL
```

13.1. а) Команда ADD дает результат 6CH, а команда AAA - 0102H.
в) Команда SUB дает результат 02H, а команда AAS не имеет эффекта.

```
13.2.          LEA  SI,UNPAK       ;Инициализация адреса
        MOV  CX,04          ; и счетчика циклов
      B20:
        OR   [SI],3OH       ;Включить тройку кода ASCII
        INC  SI             ;Перейти к следующему байту
        LOOP B20            ;Циклить 4 раза
```

13.3. В качестве примера см.рис.13.2, отличие - CX = 03.

13.4. В качестве примера см.рис.13.3, отличие - CX = 03.

13.5. а) Преобразование числа в ASCII-коде в двоичное представление:

```
                    Десятичные   Шестнадцатиричные

          8x1 =           8                 8
          2 x 10 =        20                14
          3 x 100 =       300               12C
          6 x 1000 =      6000              1770
          4 x 10000 =     40000             9C40
                                        ----
                                        B4F8
```

```
14.1.     WKDAYS    DB        'Sunday...'
               DB        'Monday... '
               DB        'Tuesday..'

14.2.     DAYNO     DB     ?
        DAYNAM    DB     9 DUP(?)
        NINE      DB     9
               LEA    SI,WKDAYS    ;Загрузить адрес таблицы
               SUB    AH,AH        ;Очистить AH
               MOV    AL,DAYNO     ;Загрузить день недели
               DEC    AL           ;-1
               MUL    NINE         ;Получить смещение
               ADD    SI,AX        ;Плюс адрес таблицы
               MOV    CX,09        ;Девять символов
               LEA    DI,DAYNAM    ;Адрес пересылки
               REP MOVSB           ;Переслать 9 символов

14.3. а)  ITEMNO    DB        '06','10','14','21','24'
     в)  ITPRICE   DW       9395,8225,9067,8580,1385
```

14.4. Возможная организация представлена в следующих процедурах:

Основной цикл   Вызов процедур

B1OREAD   Выдает на текст запроса и вводит номер элемента.
C10SRCH   Выполняет поиск в таблице и выдает сообщение об ошибке.
D10MOVE   Выделяет описание и цену из таблицы.
E10CONV   Преобразует количество из кода ASCII в двоичное представление.
F10CALC   Вычисляет стоимость (количество x цена).
G10CONV   Преобр. стоимость из двоичн. представления в симв. (код ASCII).
K10DISP   Выводит описание и стоимость на экран.

```
14.5.     SORTAB    DB        5 DUP(9 DUP(?))
               LEA       SI,ITDESC      ;Инициализация
               LEA       01,SORTAB      ; адреса таблицы
               MOV       CX,45          ; и числа символов
               CLD                      ;Направление слева направо
               REP MOVSB                ;Переслать таблицу
```

Приведенная подпрограмма пересылает таблицу. Для сортировки таблицы см. рис.14.6.

15.1. 512.

15.3. В каталоге первый байт имени файла должен быть установлен в E5.

15.5. Первый байт таблицы FAT содержит F8.

15.6. а) Позиции 28...31 в каталоге;
б) шестнадцатиричное число B4A записывается как 4A0B.

16.1. Все функции включают INT 21 H: а) 16H; в) 15H; д) 14H.

16.2. а) 4; б) 108 (9 секторов x 3 дорожки x 4 записи на дорожке);
в) одно обращение на сектор, т.е. всего 27.

16.3.Используйте рис.16.1 в качестве примера для создания файла и рис.13.6 для преобразования чисел из ASCII-кода в двоичное представление.

16.4. Используйте рис.16.2 в качестве примера для чтения файла и рис.13.6 для преобразования чисел из двоичного представления в ASCII-код.

16.5.а) Блок 0, запись 44; в) блок 1, запись 21.

16.6.Десятичное число 2652 равно шестнадцатиричному 0A5C, что записывается как 5C0A0000.

16.7. Функции включают INT 21H: а) 22H; в) 28H.

16.8. Поле FCBFLSZ содержит размер файла в байтах (произведение числа записей на длину записи), поле FCBRCSZ содержит длину записей. Разделите FCBFLSZ (четыре байта в регистровой паре DX:AX) на FCBRCSZ (два байта).

16.9. См. рис.16.4 в качестве примера для чтения блока.

16.10. В гл.13 показано, как преобразовать числа в ASCII-коде в двоичное представление.

16.11. Можно использовать рис.16.3 в качестве примера для прямого чтения и рис.13.6 для преобразования чисел из ASCII-кода в двоичное представление.

17.1. 01 и 06.

17.3.а)
```
        MOV  AH,3CH        ;Ф-ция создания файла
        MOV  CX,00         ;Нормальный атрибут
        LEA  DX,PATH1      ;ASCIIZ-строка
        INT  21H           ;Вызвать DOS
        JC   error         ;Переход по ошибке
        MOV  CUSTHAN,AX    ;Сохр. файловый номер
```

17.4.а)
```
        MOV  AH,3OH        ;Ф-ция открытия файла
        MOV  AL,OO         ;Только чтение
        LEA  DX,PATH1      ;ASCIIZ-строка
        INT  21H           ;Вызвать OOS
        JC   error         ;Переход по ошибке
        MOV  CUSTHAN.AX    ;Сохр. файловый номер
```

17.5. При открытии большого числа файлов.

18.1. Используйте INT 13H и AH = 00.

18.2. Используйте INT 13H и AH = 01.

18.4.
```
        MOV  AH,03         ;Функция записи
        MOV  AL,03         ;3 сектора
```

```
        LEA  BX,OUTDSK      ;0бласть вывода
        MOV  CH,08          ;Дорожка 08
        MOV  CL,01          ;Сектор 01
        MOV  OH,00          ;Сторона  0
        MOV  DL,01          ;Дисковод B
        INT  13H
```

18.5. Байт состояния в регистре AH содержит 00000011.

19.2. а)

```
        MOV  AH,05          ;Функция печати
        MOV  DL,0CH         ;Перевод страницы
        INT  21H
```

б)

```
        LEA  SI,NAMEFLD     ;Инициализ. имени
        MOV  CX,length      ; и длины
    B20:
        MOV  AH,05          ;Функция печати
        MOV  DL,[SI]        ;Символ из имени
        INT  21H            ;Вызвать DOS
        INC  SI             ;Следующий символ в имени
        LOOP B20            ;Циклить length раз
```

в) Можно закодировать символ перевод строки (0A) в начале адреса. Решение аналогично б),
д) Вывести еще символ перевода страницы (0C).

19.3.

```
        HEADNG      DB    13, 10, 15, 'Title', 12
```

19.5. Нельзя использовать регистр CX для пятикратного цикла, так как внутренний цикл печати имени уже использует этот регистр.Для регистра BX цикл будет следующим:

```
        MOV  BX,05
    C20:
        DEC  BX
        JNZ  C20
```

20.1. а) .SAIL.

20.2. а)

```
    MULTBY MACRO MULTPR,MULTCD
          MOV    AL,MULTCD
          MUL    MULTPR
          ENDM
```

20.3. Для включения библиотеки макрокоманд только в первом проходе необходимо кодировать:

```
    IF1
       INCLUDE library-name
    ENDIF
```

20.4. Макроопределение должно начинаться следующим образом:

```
        BIPRINT MACRO  PRTLINE,PRLEN
```

PRTLINE и PRLEN являются фиктивными аргументами для адреса и длины строки, которая предназначена для печати. Для печати с помощью BIOS INT 17H см. гл.19.

20.5.Проверку IF нельзя использовать для тестирования на нулевой делитель. Проверка IF работает только в процессе ассемблирования, в то время как тестирование на нулевой делитель должно происходить при

выполнении программы. Используйте, например, следующие команды:

```
CMP    DIVISOR,00        ;Нулевой делитель?
JNZ    (обойти)          ;Нет - обойти
CALL   (подпрограмма вывода сообщения об ошибке)
```

21.1. a) EXTRN SUBPRO:FAR

21.2. a) PUBLIC QTY,VALUE.PRICE

21.3. Для примера используйте рис.21.5.

21.4. В качестве примера передачи параметров используйте рис.21.5. Отличие состоит в том, что в данном случае используется запись в Стек трех переменных. Поэтому вызываемая программа для доступа к третьему элементу (PRICE) в стеке использует [BP+10]. Можно определить собственный стандарт для возврата элемента PRICE через стек. Обратите внимание также на операнд в команде возврата RET.

21.5. Данная программа включает материал из следующих глав: 8 (ввод-вывод на экран), 13 (преобразование между ASCII- и двоичным представлением), 12 (двоичное умножение) и 21 (связь подпрограмм). Будьте особенно внимательны при работе со стеком.

Редактировал Ершов В.Г. Дата последней редакции: 02/04/99